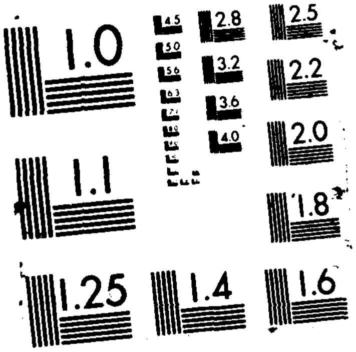
AD-A185 758   OPTIMIZATION ON ENHANCEMENT OF THE PLEXSYS NETWORK   1/3
MESSAGING SYSTEM (EMAIL)(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH G D MCALUM 1987
UNCLASSIFIED   AFIT/CI/NR-87-76T   F/G 25/5   NL


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

AD-A185 758

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

1

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
AFIT/CI/NR 87- 76T

2. GOVT ACCESSION NO.
A185 758

3. RECIPIENT'S CATALOG NUMBER

4. TITLE *(and Subtitle)*
Optimization and Enhancement of the Plexsys Network Messaging System (EMAIL)

5. TYPE OF REPORT & PERIOD COVERED
THESIS/~~DISSERTATION~~

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR*(s)*
Gary D. McAlum

8. CONTRACT OR GRANT NUMBER*(s)*

9. PERFORMING ORGANIZATION NAME AND ADDRESS
AFIT STUDENT AT: The University of Arizona

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS
AFIT/NR
WPAFB OH 45433-6583

12. REPORT DATE
1987

13. NUMBER OF PAGES
?

14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*

15. SECURITY CLASS. *(of this report)*
UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DTIC
S ELECTE D
NOV 0 4 1987
CoD

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES
APPROVED FOR PUBLIC RELEASE: IAW AFR 190-1

Lynn Wolaver

LYNN E. WOLAVER 23 Sep 87
Dean for Research and
Professional Development
AFIT/NR

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*
ATTACHED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

4

ABSTRACT

OPTIMIZATION AND ENHANCEMENT OF THE PLEXSYS NETWORK MESSAGING SYSTEM (EMAIL)

by

Gary D. McAlum

Chairman: Dr. Douglas R. Vogel

The Plexsys Network Messaging System (EMAIL) is a computer-based tool that allows users of a network system to communicate with other users through an electronic mail system. EMAIL is a memory-resident network messaging system designed to be used within a planning room environment utilizing a network host to support remote user terminals. The system can be activated by any node on the network to communicate with either another node or with all nodes on the network.

The masters report is divided into three parts: Part I is the User Guide, Part II is the Facilitator User Guide, and Part III is the System Programmer Guide. All program listings associated with EMAIL appear in appendices at the end of the report. (Keywords: [illegible] Messages; [illegible]) [illegible]

Accesion For
NTIS CRA&I ✓
DTIC TAB □
Unannounced □
Justification
By
Distribution
Availability Codes
Dist
A-1

DTIC
COPY
INSPECTED
6

87 10 20 160

OPTIMIZATION AND ENHANCEMENT
OF THE PLEXSYS
NETWORK MESSAGING SYSTEM (EMAIL)

by

Gary D. McAlum

A Report Submitted in Partial Fulfillment
of the Requirement for the Degree
of Master of Science
(Management Information Systems)
in The University of Arizona
1987

Master Committee:
Dr. Douglas R. Vogel, Chairman
Kimlynn Middleton
Bill Saints

PART I

USER GUIDE

for

THE NETWORK MESSAGING SYSTEM (EMAIL)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

PURPOSE

This document is intended to be used by personnel who will be actually using the EMAIL system. It does not explain any of the design specifications. This user guide will list the basic hardware/software requirements to run the EMAIL system. It is assumed that the EMAIL system has already been installed and is ready for use. The System Programmer Guide and the Facilitator User Guide discuss in greater detail design specifications and installation procedures.

INTRODUCTION

This document is a guide to the use of the Network Messaging System (EMAIL), a computer-based tool which allows users of the network to communicate with other individuals through an electronic mail system. EMAIL is a package of software programs and procedures designed to enable users of the network to communicate with other users during such sessions as an Electronic Brainstorming Session (EBS). These type of sessions are conducted on a network of personal computer workstations linked together on a local area network (LAN) and controlled by another personal computer called a server. The objective of the EMAIL system is to provide the opportunity for various users to communicate privately or publicly with other users such that the interchange of ideas, information, concerns, etc., is stimulated for the ultimate benefit of the group's objectives.

With the EMAIL system, communication between users is conducted through the use of individual workstations. Each participant is provided with the EMAIL system as a resident background process as well as two on-line .COM programs. The EMAIL system main menu can be entered or exited at any point without disturbing the user's current screen display. The user is informed of "new mail" by both audio and visual means. Also, the ability to determine who else is logged onto the network is provided by accessing an on-line USERNAME LIST. The EMAIL system is extremely user-friendly in that it is almost totally menu driven with clear message prompts. A user can "broadcast" a message to all users on the system, send individual mail messages, or utilize the SEND.COM function (from the DOS prompt) to send a short "screen note". Mail messages are automatically saved but can be deleted by the user if desired.

The main advantage of the EMAIL system is that it allows the private (or public) communication between users as a support tool to the primary session in progress. The person who initializes the EMAIL system, the facilitator, also has the option of keeping a log of all mail messages sent between users. This log allows for later analysis or research of session effectiveness. Participants will not be aware of the log being kept unless the facilitator informs them.

HARDWARE AND SOFTWARE REQUIREMENTS

The EMAIL system software is run on a local area network (LAN). In this environment there are two machine types, individual workstations and a server machine which controls the individual workstations. The general hardware and software requirements for operating the EMAIL system are listed in tables 2, 3, 4, and 5. Keep in mind these are minimum requirements. Performance can be improved with more advanced computers.

INSTALLATION INSTRUCTIONS

Individual users will not normally be required to install the EMAIL system on their individual workstations. There are two ways in which the EMAIL system can be initialized at user workstations. The first way is the most common. In this method the facilitator prepares and initializes the EMAIL system from the server's Central Control Menu located in the C:\SESSMAN directory. This relieves the user from any action, other than logging a username onto the network. The facilitator, when initializing the EMAIL system from the server, does not have the capability to initialize individual workstations. There are times, such as a late arrival to a session, where a workstation may need to be initialized. The following paragraph provides the input requirements for EMAIL system initialization at an individual workstation on the network.

First, ensure that the EMAIL system is not already loaded into memory. This can be easily checked by pressing a SHIFT/F7. If the EMAIL main menu comes up on the screen, then the only activity which the new or late user needs to accomplish is logging a username onto the system. From the DOS prompt, simply type:

```
LOGON <ENTER>
```

This will invoke the program which logs the user onto the system. It will erase any previous username that was assigned to that workstation (figure 1).

If the EMAIL main menu did not come up on the screen then the entire system must be loaded into memory. From the DOS prompt, simply type:

```
GOMAIL <ENTER>
```

This command allows the user to logon and loads the EMAIL system into memory. A message will be displayed on the screen when the system is ready for use, see figure 2.

OPERATING INSTRUCTIONS

The following list of items provides information on each EMAIL popup screen and the options available within each. Where appropriate, an explanation is given of the program control. This section assumes that the SHIFT/F7 key sequence has already been entered by the user.

1. EMAIL Main Menu (figure 3)

Keyboard Entry - use the up/down and/or arrow keys to select one of the following options:

-> Send Messages

** passes control to user list question menu.

-> View Received Messages

** passes control to the view messages screen.

-> Return to previous screen

** passes control back to the process which was running before the EMAIL main menu was invoked.

The user can also press the ESCAPE key to exit the EMAIL menu. Program control returns to the previous screen display.

2. User List Question Screen (figure 4)

Keyboard Entry - press "y" to display a list of usernames that are currently logged on the system.

** passes control to the username list.

Keyboard Entry - press "n" to continue on to the next screen (figure 6).

** passes control to name entry screen.

3. Username List Screen (figure 5)

Keyboard Entry - press the "ESCAPE" key to exit this screen.

** passes control to name entry screen (figure 6).

Keyboard Entry - use up/down arrows to select the name of the person you are sending to or "BROADCAST" to send to all users.

** passes control to the screen editor.

4. Name Entry Screen (figure 6)

Keyboard Entry - enter username of the person to whom you are sending the mail message or enter "*" to broadcast.

** passes control to the screen editor.

Keyboard Entry - press <ENTER> to exit this screen.

** passes control to main menu.

5. Screen Editor (figure 7)

Keyboard Entry - type in the desired mail message.

Keyboard Entry - press F1 to display the help screen.

Keyboard Entry - press F10 to send the mail message.

** control passes to main menu.

Keyboard Entry - press ALT/F9 to cancel the message.

** control passes to main menu.

Keyboard Entry - press INSERT key to toggle between insert and overwrite modes.

6. View Received Messages Screen (figure 8)

Keyboard Entry - press F2 to display next new waiting mail message.

Keyboard Entry - press F3 to reply to the mail message you are currently viewing.

** control passes to the screen editor.

Keyboard Entry - press F4 to forward the mail message you are currently viewing.

** control passes to username question screen.

Keyboard Entry - press F5 to view saved messages.

Keyboard Entry - press F6 to delete the saved message you are currently viewing.

Keyboard Entry - press the ESCAPE key to exit this screen.

** control passes to main menu.

7. Screen Notes (figures 10)

Keyboard Entry - from the DOS prompt, enter SEND to invoke the SEND.COM program (figure 10).

Keyboard Entry - from the DOS prompt, enter SEND and the username of the person you are sending to. For Example:

```
SEND JOESMITH <ENTER>
```

Keyboard Entry - from the DOS prompt, enter SEND, username of the person you are sending to, and the message you want sent. For example:

```
SEND JOESMITH THIS IS THE MESSAGE! <ENTER>
```

8. Screen Note Display Screen (figure 11)

The EMAIL system will automatically display screen notes to the user. Therefore, the user who received the SEND message only needs to indicate to the system when to remove the message from his or her workstation screen.

Keyboard Entry - press "c" to delete message from screen, see figure 11 for an example of this screen.

For a summary of the main program controls used in the EMAIL system, see table 1.

SPECIAL NOTES

There are two particular situations in which the EMAIL system will not perform as previously discussed. This is due to the safety features built into the software that prevents use of the EMAIL system under inappropriate circumstances.

The first situation will occur if the facilitator has not updated IDFILE.DAT from the Central Control Menu. This situation is recognizable when attempting to use the SEND or READ function from the main menu; program control returns to the previous screen. Until IDFILE.DAT has been updated by the facilitator the user will not be able to access the EMAIL system capabilities.

The second situation will occur when attempting to send a mail message to a user and IDFILE.DAT has been/or is being updated. If the EMAIL system has already accessed and loaded a name from a version of IDFILE.DAT which is no longer valid, any mail sent to that person will simply be disgarded. This is assuming that the recipient is not logged on in the new IDFILE.DAT.

Both of the situations described above are not typical and will rarely, if ever, occur. The primary responsibility of limiting these situations from occurring is placed on the facilitator. He or she must ensure IDFILE.DAT is updated promptly when changes are necessary. For further information on this process, see the Facilitator User Guide.

SUMMARY OF PROGRAM CONTROLS

SHIFT/F7	-	activates EMAIL main menu.
ESCAPE	-	returns control to previous screen.
INSERT	-	toggles between insert/overwrite.
ALT/F9	-	cancels message being created.
F10	-	sends created mail message.
F1	-	displays help screen in edit mode.
F2	-	displays next waiting mail message.
F3	-	allows reply to viewed message.
F4	-	forwards viewed message.
F5	-	displays saved messages.
F6	-	deletes viewed saved message.
C	-	removes screen note from display.

Table 1

SERVER MACHINE REQUIREMENTS

IBM-compatible Personal Computer
128K bytes of main storage
One 360K byte diskette drive
One 10MB hard disk drive
Monochrome or color display

Table 2

WORKSTATION MACHINE REQUIREMENTS

IBM-compatible Personal Computer
128K bytes of main storage
Two 360K diskette drives
Any color display
Any number of workstations

Table 3

SERVER SOFTWARE REQUIREMENTS

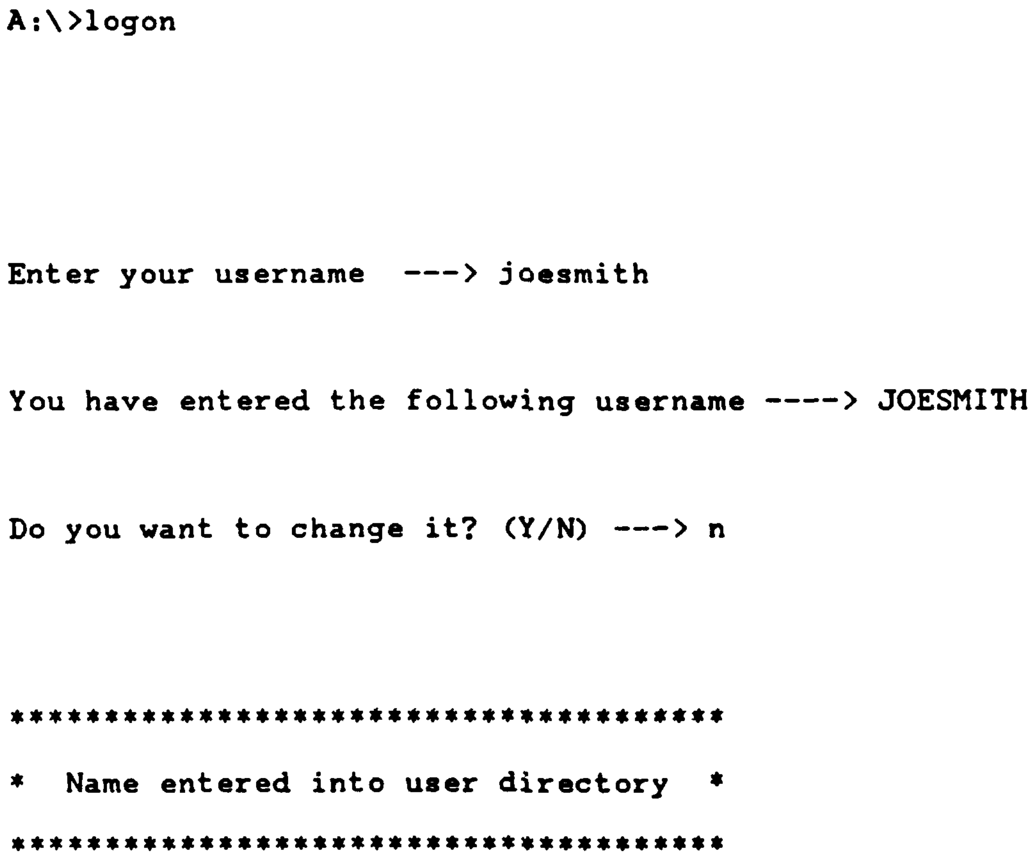
For LAN operation, a network control program, such as PC Network, is required plus MS/DOS, Release 2.1 or later.

Table 4

WORKSTATION SOFTWARE REQUIREMENTS

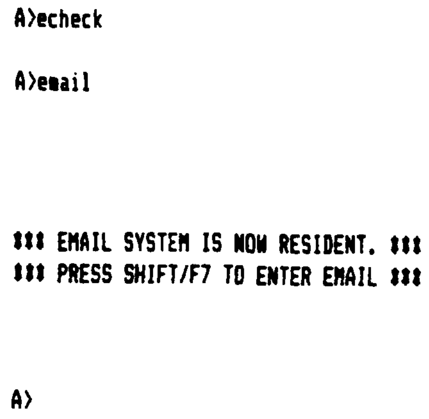
MS/DOS, release 2.1 or later is required.

Table 5



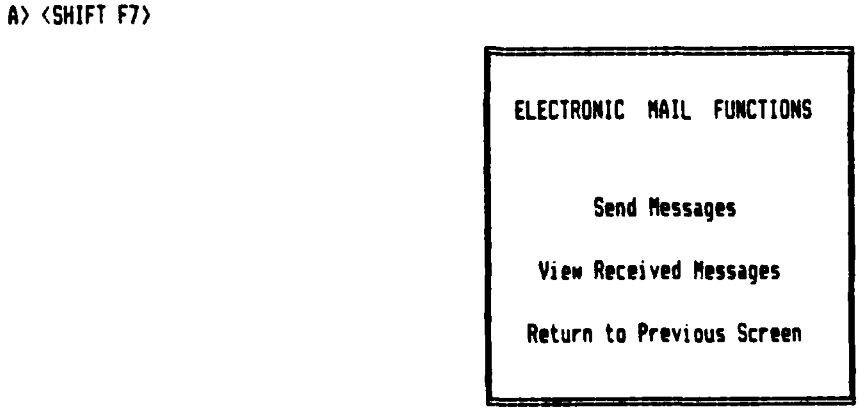
```
A:\>logon



Enter your username  ---> joesmith

You have entered the following username ----> JOESMITH

Do you want to change it? (Y/N) ---> n



**************************************
*  Name entered into user directory  *
**************************************
```

FIGURE 1



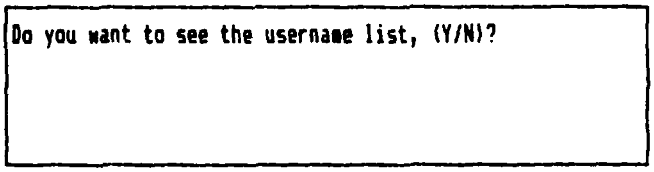
```
A>echeck

A>email



*** EMAIL SYSTEM IS NOW RESIDENT. ***
*** PRESS SHIFT/F7 TO ENTER EMAIL ***



A>
```

FIGURE 2



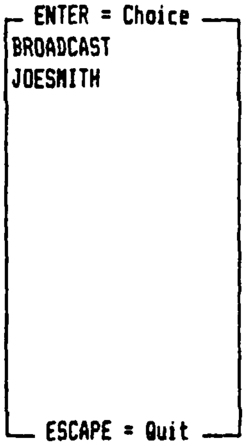
```
A> <SHIFT F7>

                         ELECTRONIC  MAIL  FUNCTIONS


                               Send Messages

                            View Received Messages

                           Return to Previous Screen
```

FIGURE 3



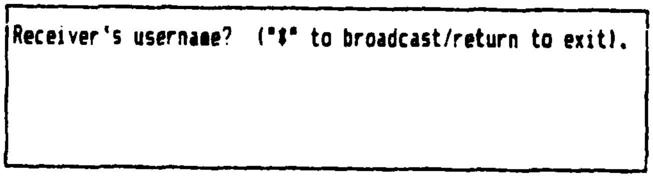
```
Do you want to see the username list, (Y/N)?
```

FIGURE 4



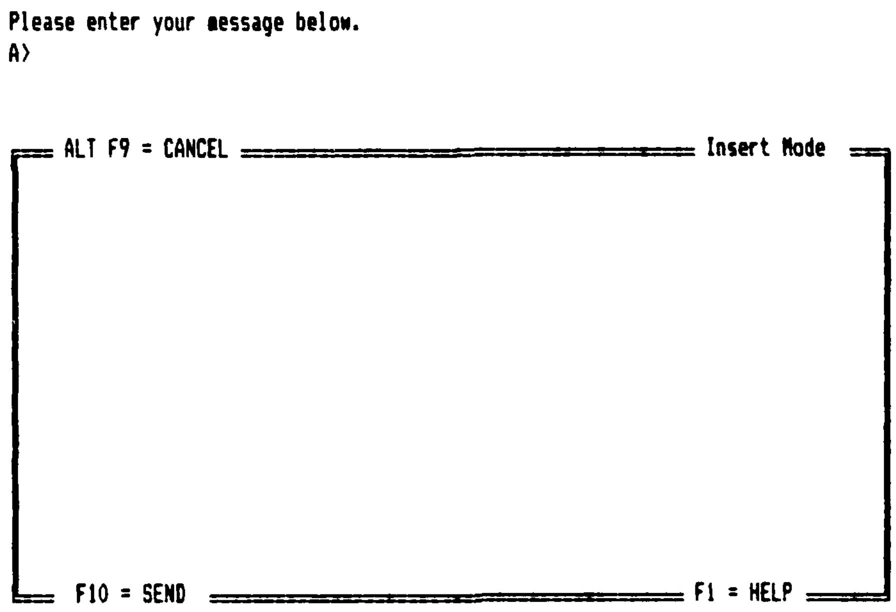
```
ENTER = Choice
BROADCAST
JOESMITH

ESCAPE = Quit
```

FIGURE 5



```
Receiver's username?  ("*" to broadcast/return to exit).
```

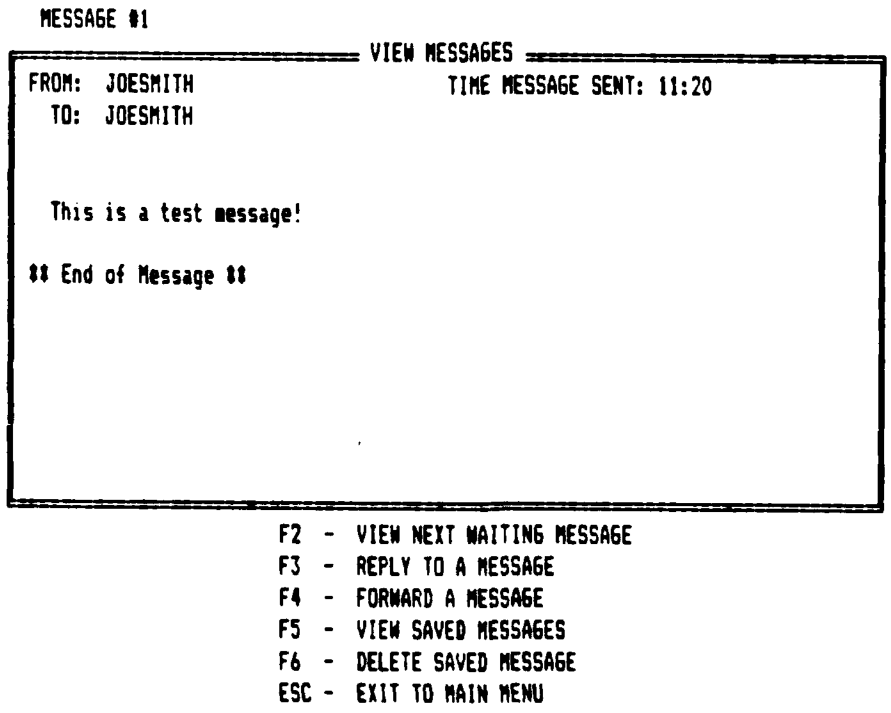
FIGURE 6



Please enter your message below.
A>

FIGURE 7

FIGURE 8



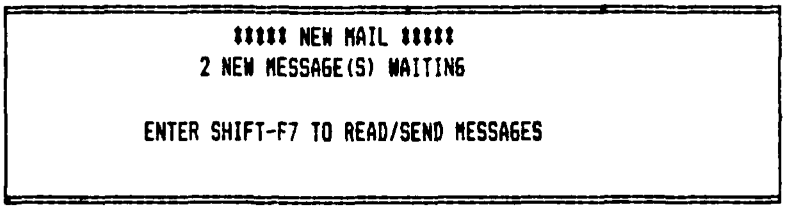
MESSAGE #1

FROM: JOESMITH TIME MESSAGE SENT: 11:20
TO: JOESMITH

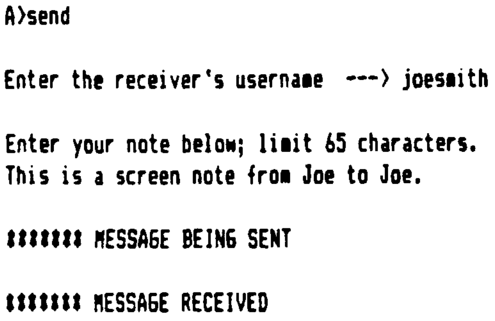
This is a test message!

End of Message

FIGURE 9



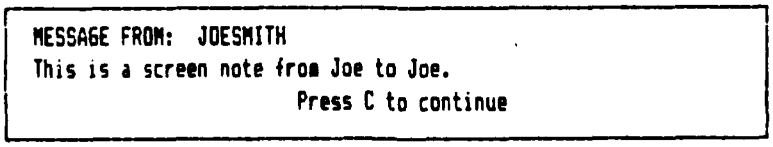
```
                  ***** NEW MAIL *****
               2 NEW MESSAGE(S) WAITING

          ENTER SHIFT-F7 TO READ/SEND MESSAGES
```



```
A>send

Enter the receiver's username  ---> joesmith

Enter your note below; limit 65 characters.
This is a screen note from Joe to Joe.

******* MESSAGE BEING SENT

******* MESSAGE RECEIVED
```

FIGURE 10

FIGURE 11



```
MESSAGE FROM:  JOESMITH
This is a screen note from Joe to Joe.
                           Press C to continue
```

PART II

FACILITATOR USER GUIDE

for

THE NETWORK MESSAGING SYSTEM (EMAIL)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

PURPOSE

This document is intended for those personnel, such as session facilitators, who need to know how to activate the EMAIL system for use. It will describe the sequence of steps necessary to initialize EMAIL for use and explain the location and types of files involved in this process. Specific design details are not discussed. Should this type of information be desired, the reader is referred to the System Programmer's Guide.

INTRODUCTION

EMAIL is a memory-resident network messaging system designed to be used within a planning room environment utilizing a network host to support remote user terminals. The system can be activated by any node on the network to communicate with either another node or with all nodes on the network.

There are two ways to send a message to another node (user). One can utilize the main EMAIL menu to send a mail message of any size, or the SEND.COM function can be used. The SEND function, executed from the DOS prompt, allows a user to send a short "note" of up to 65 characters to another user. The recipient will have this message displayed automatically on his or her screen.

With the typical mail function, a message is sent and the recipient is notified there is new mail and that the EMAIL menu must be entered to read it. EMAIL messages are automatically saved but can be manually deleted by a user. The user enters the EMAIL main menu by pressing a selected key sequence (SHIFT/F7). Once the user has finished utilizing the EMAIL functions, he/she returns to the saved state of their previously active program: editing, electronic brainstorming, etc. An on-line user list is available that allows the user to see who is logged onto the system. The user selects a username from this list for sending mail messages. When a user receives a new mail message, he/she is notified by means of a popup window and an audible signal. The recipient must enter EMAIL (SHIFT/F7) to read the mail message.

The entire EMAIL system is table driven and based upon the users currently logged onto the network. Off-line utility programs are provided at the central server to build the user table and initialize the network environment for the EMAIL system. An optional log may be selected by the facilitator if desired. This log keeps a copy of all mail messages sent and can be used for later research analysis.

HARDWARE AND SOFTWARE REQUIREMENTS

The EMAIL system software is run on a local area network (LAN). In this environment there are two machine types, individual workstations and a server machine which controls the individual workstations. The general hardware and software requirements for operating the EMAIL system are listed in tables 5, 6, 7, and 8 at the end of this document. Keep in mind these are minimum requirements.

Once the LAN is activated in a hardware/software configuration as previously described, the EMAIL system can be initialized. The necessary files and their locations are described in tables 1, 2, 3, and 4. As a facilitator you should never have to worry about installing these files. They should already be in place. They are described to give you a better idea of the EMAIL system and to assist you in any troubleshooting that may be necessary.

Each User's A Drive

LOGON.COM	- is used to log user onto the network.
SEND.COM	- allows a user to send screen notes.
EMAIL.COM	- handles the primary mail functions.
ECHECK.COM	- handles all checking and notification.
GOMAIL	- logs user on, and loads EMAIL system.

Table 1

The Server's PUBLIC Directory

EINIT.BAT	- initializes ID file/cleans out directories.
BLDFILE.COM	- initializes IDFILE.DAT with blank usernames.
FLAG.FIL	- used to "Lock" and "Unlock" the user file
GOFILE.DAT	- copies IDFILE.DAT to each MAIL directory.
EMAIL.DUM	- used to initialize the log file.

Table 2

Each User Directory

GOMAIL.BAT	- loads the EMAIL system at each workstation.

Table 3

Each User's MAIL Subdirectory

M_MENU.DAT	- contains the main EMAIL menu information.
USER.ID	- contains the user's id number.
USER.DUM	- initializes mail message files.
LOCK.COM	- prevents simultaneous access of IDFILE.DAT.
UNLOCK.COM	- allows access to IDFILE.DAT.
FLAG.FIL	- controls access to IDFILE.DAT.

Table 4

EMAIL SYSTEM INITIALIZATION

Initialization of the EMAIL system via the server is the first and most important step for the facilitator. All files needed by the system are assumed to be in place on the server and each user's disk. Assuming the network has already been activated, the EMAIL system is initialized from the server by the following steps.

1. Bring up the Central Control Menu by typing "GOPLEX".
2. Choose the entry "Activate Group-Support Tools Menu".
3. This will bring up the Group-Support Tools menu; choose the entry "EMAIL".
4. The Individual Tools Menu will be displayed which has 4 entries as in figure 1.

```
--------------------------------------------------

    EMAIL Clean up/Initialize  (log not set)
    EMAIL Clean up/Initialize  (log set)
    EMAIL System Startup
    Copy IDFILE.DAT to User Mail Directories

--------------------------------------------------
```

Figure 1

If you do not want a log kept, choose the first of the above entries from the menu. If a log is desired, choose the second of the above entries. After your choice has been entered, the system will clean out all old messages and notes from each user's directory/subdirectory. Also, IDFILE.DAT is created and initialized with blank names. This file is also placed in each user MAIL subdirectory. Finally, the system time file is established which is used for timestamping mail messages. The entire process takes about 15 seconds. At the end you will see a message similar to the figure 2.

```
*********************************
*                               *
*  EMAIL Initialization Complete *
*                               *
*********************************
```

Figure 2

The Individual Tools Menu will then be displayed again with the same 4 EMAIL entries as previously described in figure 1.

5. You are now ready to bring up the EMAIL system at each user's station. Choose the entry "EMAIL System Startup". This action will then run a .BAT file at each network station which first makes the user log a valid and correct username into IDFILE.DAT. Once a user has successfully entered a username, they will see a message displayed on their screen similar to figure 3.

```
**************************************
*  Name entered into user directory  *
**************************************
```

Figure 3

The .BAT file will then load ECHECK.COM and EMAIL.COM into resident memory. At this point a final message will be displayed on each user's screen:

```
*** EMAIL SYSTEM IS NOW RESIDENT. ***
*** PRESS SHIFT/F7 TO ENTER EMAIL ***
```

Figure 4

A key point to remember is that LOGON.COM will not permit the EMAIL system to be loaded into memory at a workstation until the user has successfully logged on. No input, other than the username, is required of the user in order to initialize the EMAIL system at their workstation.

6. The final step once all users have logged on is to update IDFILE.DAT in each user MAIL subdirectory with the current version in the PUBLIC directory. Once you are in the EMAIL menu, choose the entry which is similar to figure 5 below.

```
----------------------------------------------------

       Copy IDFILE.DAT to User Mail Directories

----------------------------------------------------
```

Figure 5

This entry will activate a file in the PUBLIC directory called GOFILE.BAT which copies the IDFILE.DAT from the PUBLIC directory into each user MAIL subdirectory. It is important to remember to perform this last task after all users are logged on, otherwise, the EMAIL system will not function correctly. In fact, because each user MAIL subdirectory is initialized with a blank version of IDFILE.DAT, no one will be able to use the EMAIL system at all until the new IDFILE.DAT has been placed in their directories. The EMAIL software is designed to access IDFILE.DAT in each user MAIL subdirectory to maximize performance. The software involved in logging a user onto the network accesses the IDFILE.DAT file in the PUBLIC directory.

SPECIAL NOTES

The advantage of having IDFILE.DAT located in the server's public directory is that it can be updated at any point in a session without disrupting other users. What this means is that a person can come in at the middle of a session and log on without impacting the session. Should a person arrive late, or a change of position is required, you do not need to reinitialize the EMAIL system from the Central Control Menu again. But you will need to enter the Central Control Menu to have IDFILE.DAT copied to each user MAIL subdirectory. There are two possible cases.

1) If the EMAIL system has already been loaded into memory at the workstation simply have the user type "logon" at the DOS prompt. This will then ensure the new username for that workstation is loaded into IDFILE.DAT. (If you are not sure whether the EMAIL system is already loaded simply enter a SHIFT/F7 to see if the main menu appears).

2) If the EMAIL system is not already loaded into memory type (or have the user type) the following command at the DOS prompt:

```
A> GOMAIL
```

GOMAIL.BAT will log the user's name into IDFILE.DAT and once this has been successfully done load the EMAIL system into memory. Remember, if you're not sure whether the EMAIL system is loaded in at a particular workstation enter a SHIFT/F7 to see if the main menu appears. Should you not check, there is the possibility of having more than one copy of the EMAIL system software running in memory. This is definitely not recommended!

In both of the above cases, bring up the Individual Tools Menu and choose the entry corresponding to figure 5. The process of copying this file to all the user MAIL subdirectories takes about 30 seconds. However, this process is transparent to the user and will not affect anyone using the system. The two programs in each of the user MAIL subdirectories, Lock and Unlock, are utilized to prevent interference with the EMAIL system while copying IDFILE.DAT to all user MAIL subdirectories.

The EMAIL system has built-in protection features which prevent the EMAIL system from being inadvertently used if the facilitator has not properly initialized the system. There are two possible situations which could occur.

The first situation is that the facilitator has brought up the EMAIL system at each workstation and users have all logged on. And, either the facilitator has not had IDFILE.DAT copied to each user MAIL subdirectory yet, has forgotten to do this, or the copy process is in progress. Suppose a user enters the EMAIL main menu by pressing SHIFT/F7 and attempts to send a mail message or read a mail message by choosing one of the first two choices on the menu? There is only a blank version of IDFILE.DAT in his/her user MAIL subdirectory at this point. The EMAIL system will not let the user enter into either the SEND or

READ mail function until a "good" copy of IDFILE.DAT is in their user MAIL subdirectory. As soon as the user attempts to enter the SEND or READ mail function the main menu will disappear and the user's previous screen will be displayed. This will continue until a correct version of IDFILE.DAT is in place.

The second situation is where a user is in the process of sending a mail message to a person who is, perhaps, listed in the username list but has since been replace or another person is logged on at that workstation. Since the user is still accessing information from the old version of an IDFILE.DAT he/she will not know that a new IDFILE.DAT has been placed in their directory. When this mail message is sent it will be simply disgarded. The next time the user attempts to send a mail message the new IDFILE.DAT will be in place.

SERVER MACHINE REQUIREMENTS
IBM-compatible Personal Computer
128K bytes of main storage
One 360K byte diskette drive
One 10MB hard disk drive
Monochrome or color display

Table 5

WORKSTATION MACHINE REQUIREMENTS
IBM-compatible Personal Computer
128K bytes of main storage
Two 360K diskette drives
Any color display
Any number of workstations

Table 6

SERVER SOFTWARE REQUIREMENTS
For LAN operation, a network control program, such as PC Network, is required plus MS/DOS, Release 2.1 or later.

Table 7

WORKSTATION SOFTWARE REQUIREMENTS
MS/DOS, release 2.1 or later is required.

Table 8

PART III

SYSTEM PROGRAMMER GUIDE

for

THE NETWORK MESSAGING SYSTEM (EMAIL)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

PURPOSE

This document is intended to be used by personnel who need to know: how to set up the EMAIL system for use on a network, what files are required and where they are located, and how to compile and make changes to the various EMAIL programs.

INTRODUCTION

EMAIL is a memory-resident network messaging system designed to be used within a planning room environment utilizing a network host to support remote user terminals. The system can be activated by any node on the network to communicate with either another node or with all nodes on the network.

There are two ways to send a message to another node. One can utilize the main EMAIL menu to send a mail message of any size, or the SEND function can be used. The SEND function, executed from the DOS prompt, allows a user to send a short "note" of up to 65 characters to another user. The recipient will have this message displayed automatically on his or her screen.

With the typical mail function, a message is sent and the recipient is notified there is new mail and that the EMAIL menu must be entered to read it. EMAIL messages are automatically saved but can be manually deleted by a user. The user enters the EMAIL main menu by pressing a selected key sequence (SHIFT/F7).

Once the user has finished utilizing the EMAIL functions, he/she returns to the saved state of their previously active program: editing, electronic brainstorming, etc. An on-line user list is available that allows the user to see who is logged onto the system. The user selects a username from this list for sending mail messages. When a user receives a new mail message, he/she is notified by means of a popup window and an audible signal. The recipient must enter EMAIL (SHIFT/F7) to read the mail message.

The entire EMAIL system is table driven and based upon the users currently logged onto the network. Off-line utility programs are provided at the central server to build the user table and initialize the network environment for the EMAIL system. An optional log may be selected by the facilitator if desired. This log keeps a copy of all messages sent and can be used for later research analysis.

SYSTEM SPECIFICATIONS

The entire EMAIL system is comprised of four on-line programs. The two primary programs are: ECHECK.PAS and EMAIL.PAS.

Description of ECHECK.PAS

ECHECK.PAS is the memory-resident program responsible for determining when new mail messages are received and for activating the short screen "notes" which may be sent via SEND.COM. ECHECK.PAS has an internal timer that checks for new mail/notes every 15 seconds. The timer can be changed to any desired time sequence by simply changing a constant in ECHECK.PAS. If there are new mail messages, ECHECK.PAS will notify the user by displaying a popup window and utilizing an audible signal. If there are any screen notes, ECHECK.PAS will display the note automatically and not remove it until the recipient has pressed 'C'.

Description of EMAIL.PAS

EMAIL.PAS is the other main program and is also memory-resident. It controls all the mail functions such as sending and reading mail messages. EMAIL.PAS does not use an internal timer because it can be activated anytime the selected key sequence is entered at the keyboard. Because of all the include files utilized, EMAIL.PAS is the largest of all programs involved in the EMAIL system.

The other two programs in the EMAIL system are: SEND.PAS and LOGON.PAS.

Description of SEND.PAS

SEND.PAS is utilized at the DOS prompt as a .COM file to send the screen notes previously discussed. If you are familiar with the SEND function on the VAX then you will already know how to use this version of the SEND function. They are exactly the same as far as appearances.

Description of LOGON.PAS

LOGON.PAS, as a .COM file, is used when a user is ready to enter into a network session but prior to loading the EMAIL system into memory. It is the critical point because LOGON.PAS is responsible for ensuring that the user enters a valid, unique username into the file IDFILE.DAT. This file is the primary file utilized by the EMAIL system. It is located in the server's PUBLIC directory and each user's MAIL subdirectory. It contains the mapping between user names and actual locations on the network. A blank version of this file is created as part of the initialization process and placed in each user's MAIL subdirectory.

All of the programs utilized in the EMAIL system were coded using Turbo Pascal and inline assembly language. A majority of the memory-resident shell was developed by Lane Ferris, et.al., as "The Hunter's Helper". This shell is public domain software. The current EMAIL system was designed and developed on a network of 16 nodes. It is easily expandable to 32 or more nodes with minimal effort as shall be discussed later.

HARDWARE REQUIREMENTS

The EMAIL system can be used on IBM PC's, or any appropriate compatibles, that are connected via a network server under a local area network configuration. In this environment there are two machine types, individual workstations and a server machine which controls the individual workstations. The general hardware requirements for a system to run EMAIL are listed in tables 1 and 2. Keep in mind, these are the MINIMUM requirements of each machine type.

SERVER MACHINE REQUIREMENTS
IBM-compatible Personal Computer
128K bytes of main storage
One 360K byte diskette drive
One 10MB hard disk drive
Monochrome or color display

Table 1

WORKSTATION MACHINE REQUIREMENTS
IBM-compatible Personal Computer
128K bytes of main storage
Two 360K diskette drives
Any color display
Any number of workstations

Table 2

EMAIL has two memory-resident programs that are designed to be loaded into each network node's memory thus decentralizing work from the server. The total amount of memory required for the resident portion of EMAIL is approximately 110K bytes of RAM. Any type of disk drive is compatible since neither of the memory-resident programs will access the disk drive again after being loaded into memory. However, LOGON.PAS and SEND.PAS are still located on the A drive and must be accessible at all times if full use of the EMAIL system is desired. All other files, as well as mail messages and screen notes, are stored on the server in various directories depending on the particular file. For example, all mail and note files are stored in each user's directory in a subdirectory called MAIL. This subdirectory also contains the EMAIL menu data file, user id file, and another file used by EMAIL.PAS. If a log is being kept, the server's public directory will contain this file.

SOFTWARE REQUIREMENTS

For local area network (LAN) operation, a network control program, such as IBM's PC Network, is required plus MS/DOS, Release 2.1 or later. To run the EMAIL system on a network already executing the appropriate network software system there are four locations where certain files must exist. These locations and files are described in tables 3, 4, 5, and 6.

Each User's A Drive

LOGON.COM	-	is used to log user onto the network.
SEND.COM	-	allows a user to send screen notes.
EMAIL.COM	-	handles the primary mail functions.
ECHECK.COM	-	handles all checking and notification.
GOMAIL.BAT	-	logs user on, and loads EMAIL system.

Table 3

The Server's PUBLIC Directory

EINIT.BAT	-	initializes ID file/cleans out directories.
BLDFILE.COM	-	initializes IDFILE.DAT with blank usernames.
FLAG.FIL	-	used to "Lock" and "Unlock" the user file
GOFILE.BAT	-	copies IDFILE.DAT to each MAIL directory.
EMAIL.DUM	-	used to initialize the log file.

Table 4

Each User Directory

GOMAIL.BAT	-	loads the EMAIL system at each workstation.

Table 5

Each User's MAIL Subdirectory

M_MENU.DAT	-	contains the main EMAIL menu information.
USER.ID	-	contains the user's id number.
USER.DUM	-	initializes mail message files.
LOCK.COM	-	prevents simultaneous access of IDFILE.DAT.
UNLOCK.COM	-	allows access to IDFILE.DAT.
FLAG.FIL	-	controls access to IDFILE.DAT.

Table 6

As previously mentioned, the EMAIL system is designed to be initialized from the server. The main menu involved is the Central Control Menu which calls the Individual Tools Menu for EMAIL. This menu is accessed by typing in "U" <ENTER> from the server's SESSMAN directory, or "GOPLEX" <ENTER> at the C:\ root directory. Specific details on how to initialize the EMAIL system are available in the Facilitator User Guide. However, should you want or need to change the EMAIL menu entries which are displayed in the Individual Tools Menu, simply edit the file EMAIL.LIS. This file is located in the SESSMAN directory.

The previous information summarizes the files necessary to run the EMAIL system on a network. To do any development work on any of the EMAIL programs you need to set up two disks with the appropriate files as described in tables 7 and 8. Batch files such as GOFILE.BAT and EINIT.BAT are not listed in either table. These can be copied to either disk if changes are necessary. However, keep in mind the space limitations of both disks.

Development Work Files: A Drive

TURBO.COM	-	the Turbo Pascal compiler.
EMAIL.PAS	-	main EMAIL functional program.
ECHECK.PAS	-	handles checking/notification.
LOGON.PAS	-	logs user onto system.
SEND.PAS	-	sends screen notes at DOS prompt.
BLDFILE.PAS	-	initializes IDFILE.DAT

Table 7

Development Work Files: B Drive

STAYSUBS.420	-	contains necessary subroutines.
STAYWNDO.341	-	creates windows.
STAYI16.410	-	handles interrupt 16 calls.
STAYI13.410	-	handles interrupt 13 calls.
STAYI21.410	-	handles interrupt 21 calls.
STAYI8.420	-	handles interrupt 8 calls.
STAYI28.410	-	handles interrupt 28 calls.
STAYSAVE.420	-	saves operating system structures.
STAYRSTR.420	-	terminates and stay resident.
CLKI8.410	-	used to set up an internal timer.
BLDMSG.INC	-	builds the mail message being sent.
GET.INC	-	reads new or saved mail message.
MENU.INC	-	displays main EMAIL menu.
STRMENU.INC	-	displays username list.
LOCK.PAS	-	controls access to IDFILE.DAT.
UNLOCK.PAS	-	used to allow access to IDFILE.DAT.

Table 8

Once you have two disks with the appropriate configuration of files as described above, you can use Turbo Pascal as you normally would to make any changes. When compiling any of the Pascal programs you need to ensure a .COM file is created by entering the options menu. Additionally, when compiling ECHECK.PAS set the MAXIMUM Segment size to 275 and the MINIMUM Segment size to 275. This is done while in the options menu prior to compiling. When compiling EMAIL.PAS set the MAXIMUM Segment size to 400. The MINIMUM Segment size will already have the default value of 400. These values are necessary because they control the stack/heap size while running the EMAIL system. Should you decide to manipulate these values you may encounter a run-time FF error, meaning you ran out of memory.

DESIGN DETAILS

Only structures applicable to MS/DOS or Turbo Pascal were used. The memory-resident shell does extensive manipulation on the operating system structures. User mail messages are sequentially numbered files (ie. USER1.N1, USER1.N2, USER1.N3, ..., USER16.N1, USER16.N2, ...,) for new mail messages and for saved messages (ie. USER1.O1, USER1.O2, ..., USER16.O1, ...,). When a new mail message is created, it is added as the next sequentially numbered message in the new mail queue. Once the new mail message has been read it is automatically saved with the new extension, such as USER5.O3. The user only has the capability to delete saved messages, not new mail messages.

With regards to the screen notes using SEND.COM from the DOS prompt there is no queueing function! When the note has been created, it is placed in a temporary holding file in the user's MAIL subdirectory called TEMP.MES. SEND will check the destination directory to see if a file called USERx.MES (x = 1, 2, ..., 16) already exists. If it exists, it means another person's note is waiting to be read. This will cause a delay for the sender until that other note has been read. Theoretically, since ECHECK checks for new mail and notes every 15 seconds the worse possible situation would be where the user sends a screen note and there is another note already there. If ECHECK has just finished checking the receiver's directory the sender would be held in a 15 second loop until ECHECK comes back around and displays the waiting message. This may not seem as efficient as utilizing a queueing function but the primary traffic will be mail messages through the normal EMAIL functions. By not utilizing a queueing function, the overall complexity of the SEND function was kept to a minimum. A potential timing problem could occur if two users with sequential locations on the network send a screen note to the same destination at the exact time. One of the screen notes might be overwritten. The probablilty of this type of situation occurring is extemely low.

IDFILE.DAT

A key point to be discussed is in relation to the file which is the main interface between the EMAIL system and the local area network on which it is running. Remember that IDFILE.DAT contains the username of each user and their "location" on the network. As already explained IDFILE.DAT is accessed by all users when logging onto the network. When all users have logged on, the facilitator then has a copy of IDFILE.DAT placed in each user's MAIL subdirectory. Therefore, keep in mind that LOGON.PAS accesses IDFILE.DAT in the PUBLIC directory and EMAIL.PAS and SEND.PAS access IDFILE.DAT in the individual user MAIL subdirectories.

When logging on, IDFILE.DAT must be accessed by all users. In order to allow access by multiple users of this file (in the PUBLIC directory) a "Lock" and "Unlock" procedure has been implemented in LOGON.PAS. The Lock procedure is called before any code within a program that accesses IDFILE.DAT. It first looks in the server's public directory to see if a file called FLAG.FIL exists or not. If it does, this means that IDFILE.DAT is available for use. Procedure Lock then renames FLAG.FIL to TEMP.FIL to prevent other programs from accessing IDFILE.DAT. This action alone was not enough to prevent simultaneous "collisions" from occurring. Turbo Pascal compiler directives have been added around the code which attempts the renaming of FLAG.FIL. This prevents a run-time error from occurring should two or more programs try to rename the file. Procedure Unlock is placed after the code which accesses IDFILE.DAT to "release" it for other programs to access. Unlock simply renames TEMP.FIL back to FLAG.FIL.

Similarly, the same procedures are used in EMAIL.PAS and SEND.PAS. Although these programs access their own personal copy of IDFILE.DAT, there are times when simultaneous access of IDFILE.DAT might occur. For example, the facilitator can copy new versions of IDFILE.DAT to each user MAIL subdirectory if a person logs on late or a change of seating arrangement occurs. The batch file, GOFILE.BAT, also utilizes programs LOCK and

UNLOCK which are the same as the procedures LOCK and UNLOCK.

Special Programming Notes

Currently, ECHECK.COM is set for a 15 second interval. In other words, every 15 seconds it executes the check for new mail messages and screen notes. This time was purely arbitrary and can be adjusted by changing the variable called TIMER_TIME.

To enter the main EMAIL system menu, the user must press SHIFT/F7. This can also be easily changed if desired. A variable in EMAIL.PAS called OUR_HOTKEY contains the value corresponding to SHIFT/F7.

As mentioned before, the EMAIL system was designed and developed on a local area network of 16 nodes. However, to expand the operating capacity of the EMAIL system is quite simple. The first step is to change a constant called NODES to the desired number, such as 32, in each of the following programs: BLDFILE.PAS, LOGON.PAS, SEND.PAS, ECHECK.PAS, and EMAIL.PAS. The next step is to modify EINIT.BAT. The appropriate code for extension to 32 nodes is already present in EINIT.BAT but is commented out. Simply remove the lines preceded by REM, which designates a comment in BAT files. The final, but critical, step is to modify the user table in SEND.PAS and EMAIL.PAS. The appropriate code is already present to expand to 32 nodes but is commented out. Simply remove the comment brackets and append that code to the end of the user table. Remember, the additional nodes must have the same user directories on the server and contain the necessary files as mentioned previously.

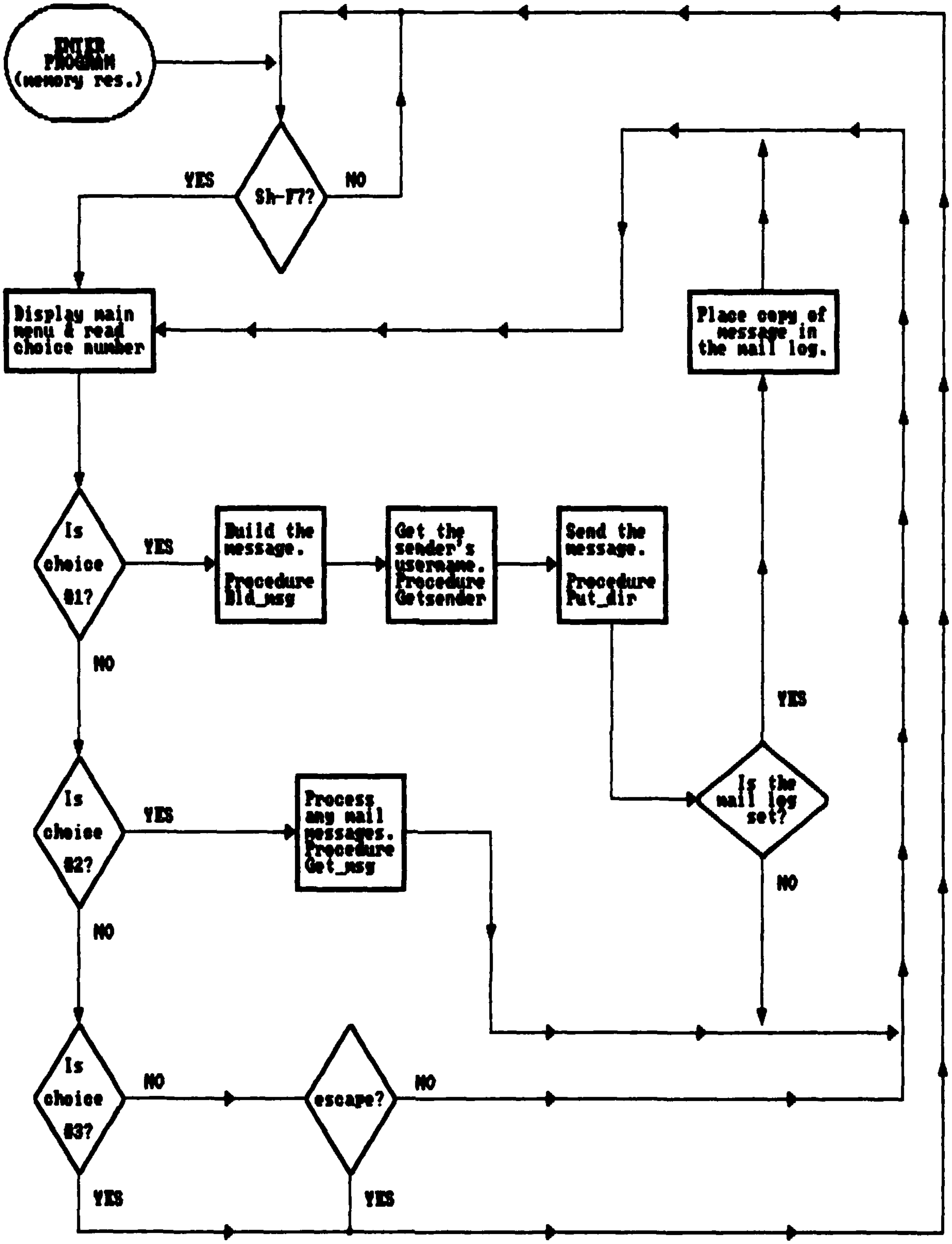
System Limitations

Once loaded the EMAIL system cannot be removed from memory except by rebooting the computer station that it is running on. If you have to reboot a particular station none of the ordinary initialization done on the 5server has to be done as this would severely disrupt the rest of the users logged on. Simply, reboot the particular machine and enter the following command.

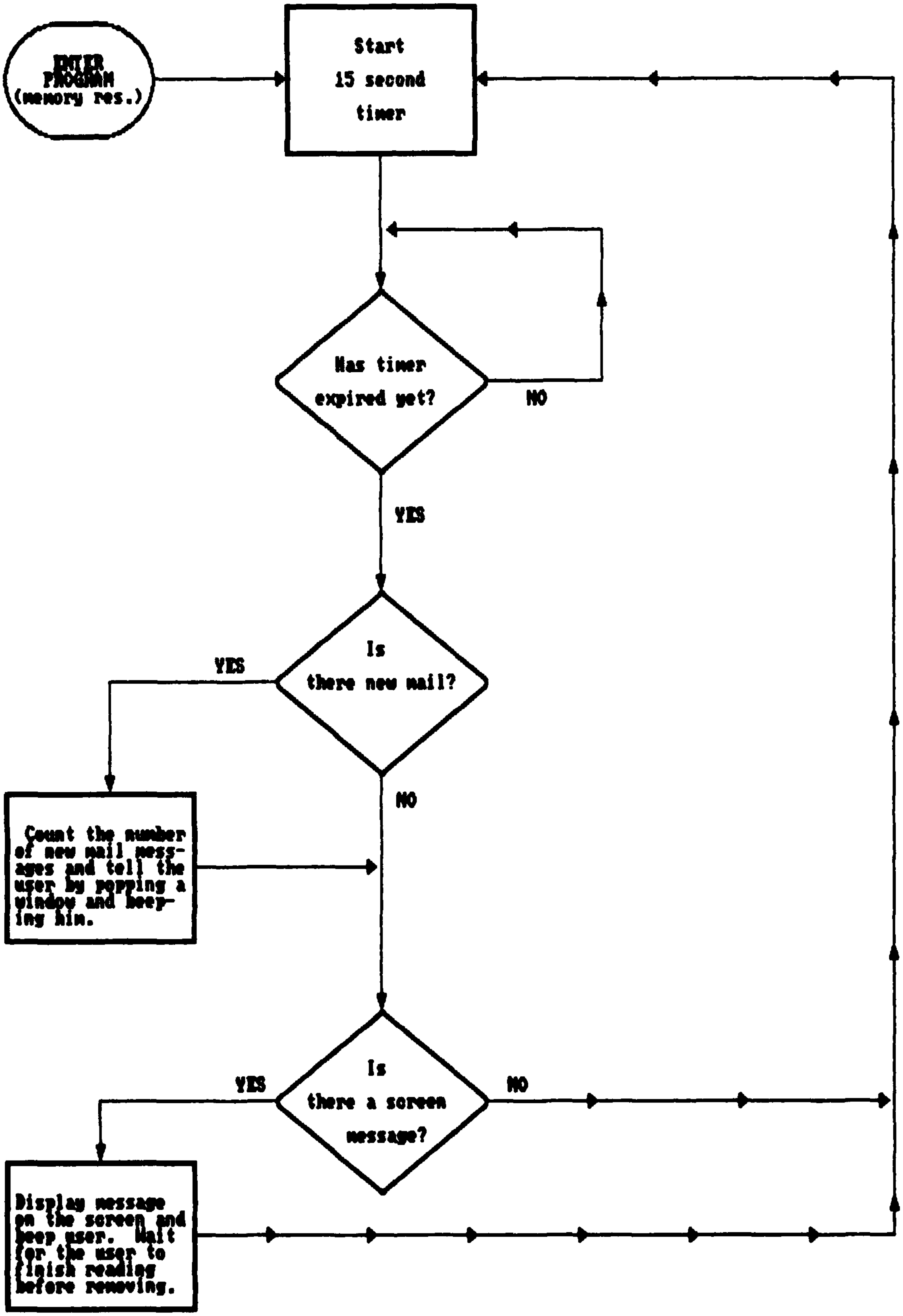
```
A> GOMAIL <ENTER>
```

Mark and Release

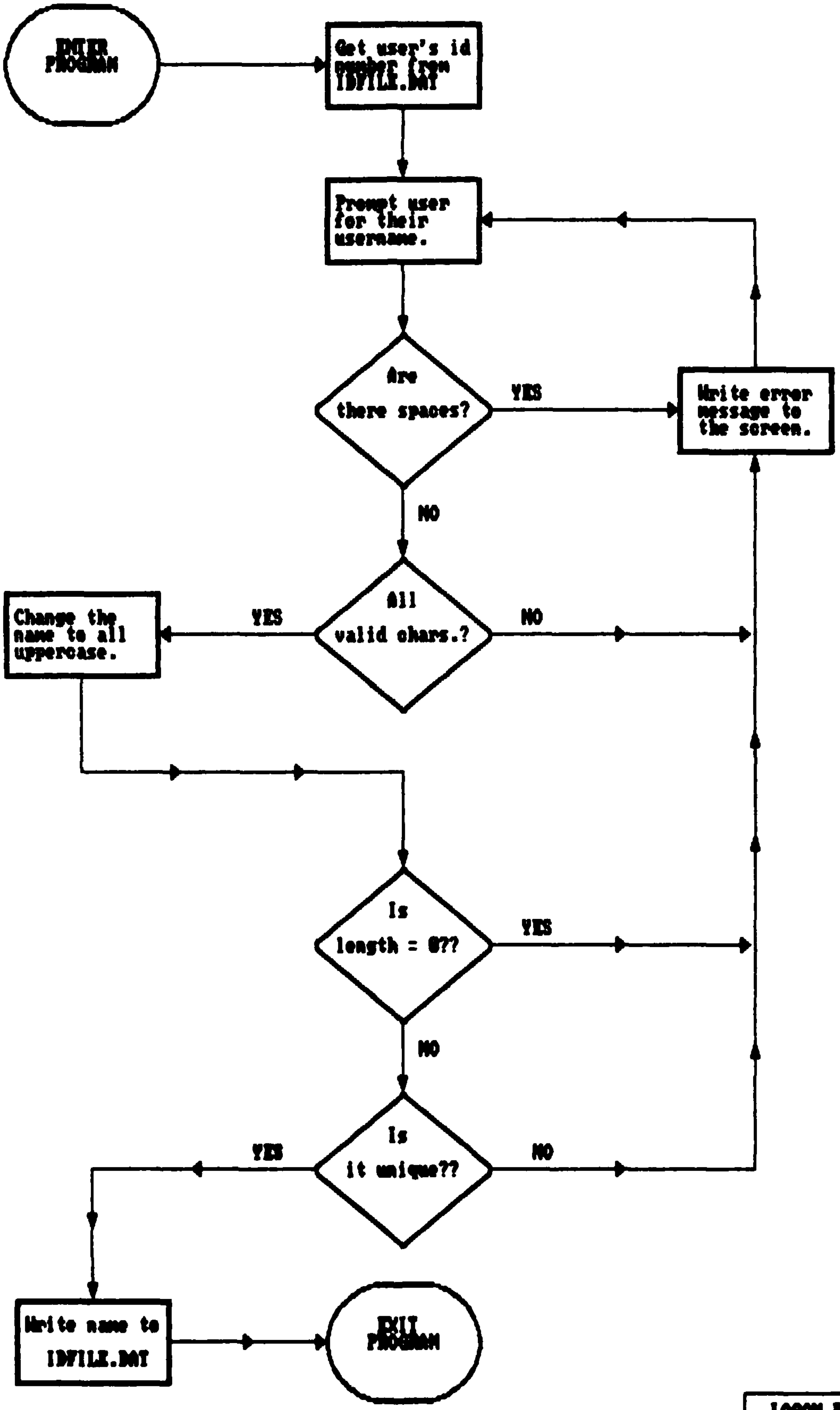
There are a couple of public domain programs which can be used to remove programs from memory. They are called MARK and RELEASE. Prior to loading ECHECK and EMAIL, you would enter MARK at the terminal. When you want to take ECHECK and EMAIL out of memory, you would enter RELEASE. A potential problem arises from the fact that when a RELEASE command is entered, any memory-resident programs loaded into memory after the last MARK command will be released. If you can always ensure that ECHECK and EMAIL are the last programs loaded after MARK then this will not present a problem. But, if this is not always the case then you should consider carefully whether or not you want to use the MARK and RELEASE commands.


ENTER PROGRAM (memory res.)
Sh-F7?
YES
NO
Display main menu & read choice number
Place copy of message in the mail log.
Is choice #1?
YES
Build the message. Procedure Bld_msg
Get the sender's username. Procedure Getsender
Send the message. Procedure Put_dir
NO
YES
Is choise #2?
YES
Process any mail messages. Procedure Get_msg
Is the mail log set?
NO
NO
Is choice #3?
NO
escape?
NO
YES
YES

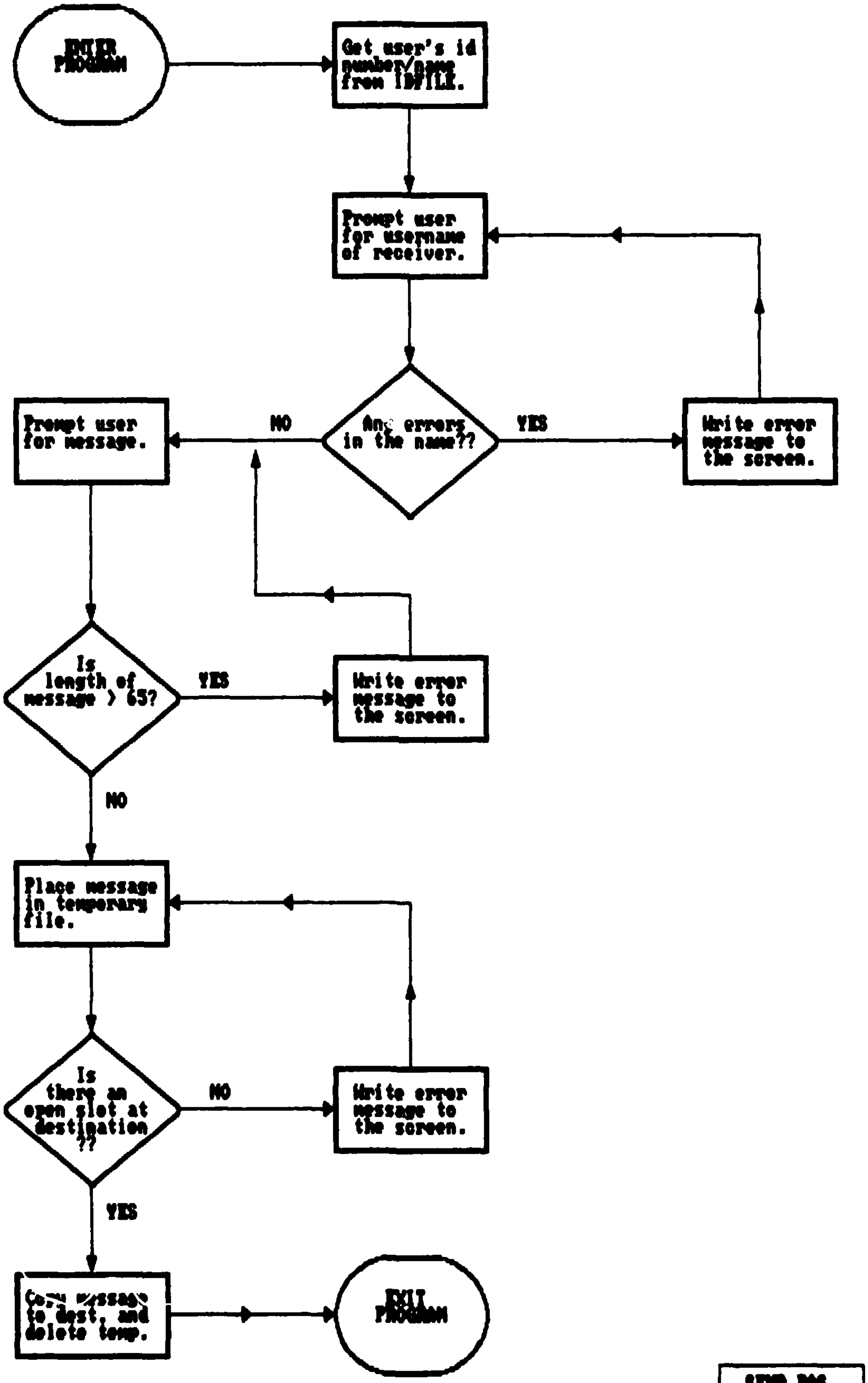
EMAIL.PAS Flow Diagram


ENTER PROGRAM (memory res.)
Start 15 second timer
Has timer expired yet?
NO
YES
Is there new mail?
YES
NO
Count the number of new mail messages and tell the user by popping a window and beeping him.
Is there a screen message?
YES
NO
Display message on the screen and beep user. Wait for the user to finish reading before removing.

ECHECK.PAS
Flow Diagram


ENTER PROGRAM
Get user's id number from IDFILE.DAT
Prompt user for their username.
Are there spaces?
YES
Write error message to the screen.
NO
All valid chars.?
YES
NO
Change the name to all uppercase.
Is length = 0??
YES
NO
Is it unique??
YES
NO
Write name to IDFILE.DAT
EXIT PROGRAM

LOGON.PAS
Flow Diagram


ENTER PROGRAM
Get user's id number/name from IDFILE.
Prompt user for username of receiver.
Any errors in the name??
NO
YES
Write error message to the screen.
Prompt user for message.
Is length of message > 65?
YES
Write error message to the screen.
NO
Place message in temporary file.
Is there an open slot at destination??
NO
Write error message to the screen.
YES
Copy message to dest. and delete temp.
EXIT PROGRAM

SEND.PAS
Flow Diagram

APPENDIX A

PROGRAM LISTING

of

EMAIL.PAS

```
{****************************************************************************
*                                                                          *
* EMAIL.PAS       GARY MCALUM               August 1987                    *
*                                                                          *
* PROGRAM DESCRIPTION:                                                     *
*                                                                          *
* EMAIL is a memory-resident network message system designed to be used   *
* within a planning room environment.  This program can be activated by    *
* any network node to send or receive messages to a specific node or all   *
* nodes active on the network.  When a user receives a message, he is      *
* notified by means of a another memory-resident program called ECHECK.    *
* EMAIL is table driven and is based upon the users currently on logged on *
* the network.  Messages sent on the system contain both the sender's and  *
* and receiver's names as well as a time stamp on when the message was     *
* sent.  An optional log may be created at initialization that allows all  *
* messages sent on the system to be logged and kept for later analysis.    *
*                                                                          *
* INPUT/OUTPUT FILES:                                                      *
*                                                                          *
* (1)  USER.ID      : contains the unique user number in ASCII (1-16)      *
* (2)  USER.DUM     : used to write a blank screen for the new message     *
* (3)  USERX.N*     : X = user#;  * = 1,2,3...;  new mail message          *
* (4)  USERX.O*     : X = user#;  * = 1,2,3...;  old mail message          *
* (5)  M_MENU.DAT   : a menu file for the main EMAIL program menu          *
* (6)  IDFILE.DAT   : the user table, created each system init             *
* (7)  TIME.FIL     : contains the base sys time, replaced each sys init   *
* (8)  FLAG.FIL     : is used by procedures Lock and Unlock; for IDFILE    *
* (9)  EMAIL.LOG    : an optional log file to store mail msg interaction   *
*                                                                          *
* PROGRAM REQUIREMENTS:                                                    *
*                                                                          *
* Memory required is approximately 82k bytes of RAM.  Any type of disk     *
* drive is compatible since the program is loaded directly in memory and   *
* does not access the disk containing the source program afterwards.       *
* EMAIL was developed and tested using MS/DOS Version 3.1 and TURBO Pascal *
* Version 3.0.  This program should be assembled as a COM file with mini-  *
* mum dynamic memory to set 0400 and max. dynamic memory set to 0400. This *
* ensures enough memory is allocated to the heap to avoid collisions while *
* limiting the amount of memory reserved for the heap.                     *
*                                                                          *
****************************************************************************}

{$R-}
{$C-}

PROGRAM email;

{*********************** CONSTANTS USED IN THE SHELL *************************}
CONST
  nodes      = 16;      { *** number of nodes on network *** }
  maxwin     = 10;      {Max # windows open at 1 time, used in window.inc}
  esc        = #27;     {character equivalent of Escape Key}
  alt_f9     = #112;    {Alt Function 9 Scan code        }
  ctrl_f9     = #229;     {Ctl-F9 (#102+127)  Key  code    }
  quit_key    = ctrl_f9; {Quit and Release Memory}
  alt         = 08;       {Shift bits at 40:17 }
  ctrl        = 04;       {the code for the control key}
  left_shift = 02;        {used to shift left}
  rght_shift = 01;        {used to shift right}
  biosi8      = 8;        {Bios Timer interrupt}
  biosi16     = $16;      {Bios Keyboard interrupt}
  biosi13     = $13;      {Bios Disk interrupt}
  dosi21      = $21;      {DOS service router interrupt}
  dosi28      = $28;      {DOS Idle interrupt}

{*********************** TYPE DECLARATIONS USED IN THE SHELL *****************}

TYPE
  entry_name     = string[20];
  entry          = record                {make IDFILE.DAT a random access file}
                     name : entry_name;
                     id   :integer;
                   end;

  regtype        = RECORD
                     ax,bx,cx,dx,bp,si,di,ds,es,flags:INTEGER
                   END;
  halfregtype    = RECORD
                     al,ah,bl,bh,cl,ch,dl,dh:Byte
                   END;
  filename_type = STRING[64];
  str_66         = STRING[66];
  user_table     = ARRAY[1..nodes] OF str_66;
  vector         = RECORD                        {Interrupt Vector type}
                     ip,cs :INTEGER ;
                   END ;
  rayptr         = ^bigray;
  bigray         = RECORD
                     nc,nl:INTEGER;
                     pixray: ARRAY[1..4000] OF Byte;
                   END;

{*********************** TYPED CONSTANTS USED IN THE SHELL *******************}

CONST
  our_hotkey   : Byte = 90;                  {scan code for SHIFT-F7}

{*********************** SCAN CODE CAN BE CHANGED TO MAKE ********************}
{*********************** ANOTHER KEY ACTIVE AS THE HOST KEY. *****************}

      { This table marks those INT 21 functions which must be passed     }
      { without modification. They either never return, fetch parameters }
      { from the stack, or may be interrupted by a TSR.                  }

      functab    : ARRAY[0..$6f] OF Byte =
                   (1,1,1,1, 1,1,1,1, 1,1,1,1, 1,0,0,0,  {0-C}
                    0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0,
                0,0,0,0, 0,0,1,0, 0,0,0,0, 0,0,0,1,  {26,2F}
                0,1,1,1, 1,1,0,0, 0,0,0,0, 0,0,0,0,  {31-35}
                0,0,0,0, 0,0,0,0, 1,1,1,1, 1,1,0,0,  {48-4D}
                1,1,1,1, 0,1,0,0, 1,0,0,0, 0,1,1,1,  {50-53,55,58,5D-5F}
                1,1,1,1, 1,1,1,1, 1,1,1,1, 1,1,1,1); {60-62}

  intr_flags  : Byte = 0;          {Active interrupts flags}
    int13_on  =  04;               {Disk  interrupt is active}
    int21_on  =  08;               {DOS Service router is active}
  status      : Byte = 0;          {Status of current TSR activity}
    hotkey_on =  01;               {Received the HotKey}
    inuse     =  02;               {TSR is active}
    foxs      = $ff;               {workaround for inline hex FF}
  dosversion  : Byte = 0;          {Current Version of DOS}
  waitcount   : Byte = 0;          {Wait to activate count}
  userprogram :INTEGER = 0;        {Offset to Users Program Code}
  ourdseg     :INTEGER = 0;        {Turbo Data Segment Value  }
  oursseg     :INTEGER = 0;        {Turbo Stack Segment Value   }
  dosdseg     :INTEGER = 0;        {Dos Datasegment value     }
  dossseg     :INTEGER = 0;        {Dos Stack Segment Value   }
  dossptr     :INTEGER = 0;        {Dos Stack pointer value   }
  dosssiz     :INTEGER = 0;        {Dos Stack size in words }
  usrdseg     :INTEGER = 0;        {Interrupted Datasegment value     }
  usrsseg     :INTEGER = 0;        {Interrupted Stack Segment Value   }
  usrsptr     :INTEGER = 0;        {Interrupted Stack pointer value   }
  ourpsp      :INTEGER = 0;


{ The following  constants #MUST# remain in the IP:CS order.   }
{ StaySave uses them as  JMP targets                           }

  bios_int8   :vector = (ip:0;cs:0);  {BIOS Timer Interrupt Vector }
  bios_int16  :vector = (ip:0;cs:0);  {BIOS Keyboard Interrupt Vector }
  bios_int13  :vector = (ip:0;cs:0);  {BIOS Disk Interrupt Vector    }
  dos_int21   :vector = (ip:0;cs:0);  {DOS Sevice Interrupt Vector}
  dos_int28   :vector = (ip:0;cs:0);  {DOS idle Service interrupt Vector}
  dosstat1    :vector = (ip:0;cs:0);  {Pointer to INDOS byte}
  dosstat2    :vector = (ip:0;cs:0);  {Pointer to CRITICAL byte}

  version       :STRING[4] = '4.15';  {current Version number}
  new_ext       :STRING[2] = '.N';    {used to designate new mail msgs}
  timer_message :STRING[80] = '';

  flag_fil    :str_66 = 'D:\MAIL\FLAG.FIL';   {used by Lock and Unlock}
  temp_fil    :str_66 = 'D:\MAIL\TEMP.FIL';   {used by Lock and Unlock}
  log_str     :str_66 = 'E:EMAIL.LOG';        {log-contains a sessions mail}
  usr_id_str  :str_66 = 'D:\MAIL\USER.ID';    {contains user's id number}
  in_fil_str  :str_66 = 'D:\MAIL\IN.FIL';     {contains sender's message}
  usr_path    :str_66 = 'D:\MAIL\USER';       {init path strg w/i ea user dir}
  idfile_dat  :str_66 = 'D:\MAIL\IDFILE.DAT'; {contains usernames/id#'s}
  m_menu_dat  :str_66 = 'D:\MAIL\M_MENU.DAT'; {contains initial menus}
```

```
      user_filetable: user_table =        {user path table used in PUT.DIR}
      ('E:\USER1\MAIL\',                  {procedure to enable file copying}
       'E:\USER2\MAIL\',                  {from one node directory to another}
       'E:\USER3\MAIL\',
       'E:\USER4\MAIL\',
       'E:\USER5\MAIL\',
       'E:\USER6\MAIL\',
       'E:\USER7\MAIL\',
       'E:\USER8\MAIL\',
       'E:\USER9\MAIL\',
       'E:\USER10\MAIL\',
       'E:\USER11\MAIL\',
       'E:\USER12\MAIL\',
       'E:\USER13\MAIL\',
       'E:\USER14\MAIL\',
       'E:\USER15\MAIL\',
       'E:\USER16\MAIL\');

{*********************************************************************}
{* To expand this program to handle 32 users on the network, simply  *}
{* "uncommert" the following code and append to the above usertable  *}
{* Remember that the variable NODES also has to be set to 32 (above).*}
{*********************************************************************}

{      'E:\USER17\MAIL\',
       'E:\USER18\MAIL\',
       'E:\USER19\MAIL\',
       'E:\USER20\MAIL\',
       'E:\USER21\MAIL\',
       'E:\USER22\MAIL\',
       'E:\USER23\MAIL\',
       'E:\USER24\MAIL\',
       'E:\USER25\MAIL\',
       'E:\USER26\MAIL\',
       'E:\USER27\MAIL\',
       'E:\USER28\MAIL\',
       'E:\USER29\MAIL\',
       'E:\USER30\MAIL\',
       'E:\USER31\MAIL\',
       'E:\USER32\MAIL\');   }

{******************************* VARIABLES *********************************}


VAR
  an_entry   : entry;            { username and node number in idfile }
  listid,                        { temporary work file }
  idlist     : file of entry;    { random access files }
  ctr        : integer;          { counter index }
  ulist      : text;             { textfile }
  xfree,                   { used by shell to release image, procedure putx }
  unlocked,                { used by Lock and Unlockl }
  skip,reply,              { flags used to determine escapes, reply, forward }
  for_ward,
  firstime   : boolean;    { used in get.inc }
  tempfile,
  flagfile,
  jurl       : text;       { temporary file used by get.inc, used for REPLY }
  n_line     : string[27]; { used to store sender's name when REPLY is called }
  tempstr    : string[20]; { stores chosen name from username list, bldmsg.inc }

  regs       : regtype;
  halfregs   : halfregtype Absolute regs;
  keychr     : CHAR ;
  bytecount  : INTEGER;
  savedpsp   : INTEGER;          { Program Segment Prefix pointers }
  error      : INTEGER;          { I/O results }
  good       : BOOLEAN;          { I/O results switch }
  terminate  : BOOLEAN;          { Exit stayRes Flag  }
  ourdta    :ARRAY [1..2] OF INTEGER; {Local DTA pointer}
  saveddta :ARRAY [1..2] OF INTEGER; {Interrupted DTA pointer}

  {NEEDED FOR MAIL PROGRAM}

  tics               :REAL;      {used to set timer}
  done,                          {flag used in bldmsg}
  broadcast,                     {used to flag if broadcast is requested}
  log_on             :BOOLEAN;   {set to true if the log option is on}
  msgnum             :STRING[2]; {message number of the mail}
  usrno              :STRING[2]; {user number}
  ctime              :STRING[5]; {character representation of the time}
  from_and_from_name :STRING[27]; {concat of FROM: and sender user name}
  to_and_to_name     :STRING[27]; {concat of TO: and receiver user name}
  heaptop            :^INTEGER;  {used to mark top of heap}
  hiclock            :INTEGER Absolute $40 :$6e;
  loclock            :INTEGER Absolute $40 :$6c;
  msgint,                        {message number in integer format}
  bt,                            {index used in broadcasting}
  send_id,                       {sender's user number}
  xp,                            {used in mail_man for xcursor position}
  yp                 :INTEGER;   {used in mail_man for ycursor position}
  user,                          {user idfile string}
  log_file,                      {email.log file string}
  infile,                        {used in putdir, designates sender's file}
  outfile            :TEXT;      {used in putdir, desig. receiver's file}
  newfile            :str_66;    {file string name, locates new mail files}
  pic                :rayptr;    {used by driver, ptr to orig user screen}

{--------------------------------------------------------------------------}
{               W  I  N  D  O  W     R  O  U  T  I  N  E                   }
{--------------------------------------------------------------------------}
{$I b:STAYWNDO.341}

{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
{--------------------------------------------------------------------------}
{              THE FOLLOWING ARE THE USER INCLUDE ROUTINES                 }
{--------------------------------------------------------------------------}
{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
```

```
{$I b:STAYSUBS.420}

{****************************************************************************
*  CLICK;  Makes a clicking sound.                                          *
****************************************************************************}
Procedure Click;

Begin
  Sound(2000);
  Delay(5);
  NoSound;
End;


{****************************************************************************
*  CURSORON;  TURNS THE CURSOR ON                                           *
****************************************************************************}
PROCEDURE cursoron(turnon: BOOLEAN);
TYPE
   reglist = RECORD
               ax,bx,cx,dx,bp,si,di,ds,es,flags: INTEGER
             END;
VAR
   reg: reglist;

BEGIN
  IF turnon THEN
    IF Mem[0:$449] = 7 THEN
      reg.cx := $0c0d
    ELSE
      reg.cx := $0607
  ELSE
    reg.cx := $2000;
  reg.ax := $0100;
  Intr($10, reg)
END;

{****************************************************************************
*  EXIST;  RETURNS TRUE IF this_file EXISTS IN THE DIRECTORY                *
*****************************************************************************}
FUNCTION exist(this_file: str_66):  BOOLEAN;

VAR
   fil : FILE;

BEGIN
  Assign(fil, this_file);
  {$I-}
  RESET(fil);
  {$I+}
  exist := (IOResult = 0);
  CLOSE(fil);
END;
```

```
{***************************************************************************}
{ This procedure is used to rename a file called FLAG.FIL to a file called }
{ TEMP.FIl.  This occurs when IDFILE.DAT is being accessed.  In effect, it }
{ "locks out" IDFILE.DAT from the other processes.  If FLAG.FIL does not   }
{ exist, signifying that IDFILE.DAT is already in use, this procedure will }
{ go into a loop until IDFILE.DAT is available to be locked out.           }
{***************************************************************************}

Procedure lock;
 var go : boolean;
 begin
  assign(tempfile,temp_fil);
  assign(flagfile,flag_fil);
  unlocked := true;
  REPEAT
   if  exist(flag_fil) then
     begin
       {$I-}                        {compiler directive, turnoff run time check}
       rename(flagfile,temp_fil);
       {$I+}                        {compiler directive, turnon run time check }
       go := (IOresult = 0);
       if go then
         begin
           close(tempfile);
           unlocked := false;
         end;
     end
   else
     begin
       delay(1000);
     end;
  UNTIL not unlocked;
 end;


{***************************************************************************}
{ This procedure is used to "unlock" IDFILE.DAT.  It does this by renaming }
{ the file TEMP.FIL to FLAG.FIL.  It also sets a flag called unlocked to   }
{ true.  This flag is used by Procedure Lock.                              }
{***************************************************************************}

Procedure unlock;
 begin
  unlocked := true;
  rename(tempfile,flag_fil);
  close(flagfile);
 end;


{****************************************************************************
*  GETX;  TAKES A PICTURE OF THE USER SCREEN                                *
****************************************************************************}
FUNCTION getx(cl,LN, ncol, nln: INTEGER): rayptr;
```

```
VAR
  i,                    {Loop control variable}
  j,                    {Loop control variable}
  numbytes,             {number of bytes}
  start      : INTEGER; {start position}
  ray        : rayptr;  {position on screen pointer}
  screen_mem : ARRAY [1..4000] OF Byte Absolute $B800:0000;

BEGIN
  NEW(ray);
  start := ((LN-1) * 160) + ((cl * 2) - 1);
  numbytes := (ncol * 2);
  j := 1;
  FOR i := 1 TO nln DO
    BEGIN
      Move(screen_mem[start],ray^.pixray[j],numbytes);
      start := start + 160;
      j := j + numbytes;
    END;
  ray^.nc := ncol;
  ray^.nl := nln;
  getx := ray;
END;

{******************************************************************************
*  PUTX;  RETURNS PICTURE OF THE USER SCREEN BUT SAVES THE IMAGE             *
*******************************************************************************}
PROCEDURE putx(cl,LN: INTEGER; ray: rayptr);
VAR
  i,                    {Loop control variable}
  j,                    {Loop control variable}
  numbytes,             {number of bytes}
  start       :INTEGER; {start position}
  screen_mem2 : ARRAY[1..4000] OF Byte Absolute $B800:0000;

BEGIN
  start := ((LN-1) * 160) + ((cl * 2) - 1);
  numbytes := (ray^.nc * 2);
  j := 1;
  FOR i := 1 TO ray^.nl DO
    BEGIN
      Move(ray^.pixray[j],screen_mem2[start],numbytes);
      start := start + 160;
      j := j + numbytes;
    END;
  if xfree then         { this signals PUTXFREE was desired }
       dispose(ray);
END;


{-----------------------------------------------------------------------------}
{                     NOW BEGINS THE REAL PROGRAM                              }
{-----------------------------------------------------------------------------}
```

```
{#############################################################################
#  DRIVER;                                                                   #
##############################################################################}
PROCEDURE driver;
CONST
  twyblks             = '                    ';
  user_ext: STRING[4] = 'user';
  old_ext:  STRING[2] = '.0';             {extension meaning an old file}
  dum_ext:  STRING[4] = '.dum';
  send_ext: STRING[6] = 'IN.FIL';         {name of the send msg file}
  top_of_file         = 'TOP OF FILE! ';  {warning message}
  end_of_file         = 'END OF FILE! ';  {warning message}
  from                = 'FROM:  ';
  to_u                = '  TO:  ';
  time_file           = 'D:time.fil';     {contains the base time}
  blanks20            = '                    ';
  no_border           = 0;   {indicates no border}
  no                  = 0;   {no for an operation}
  yes                 = 1;   {yes for an operation}
  frame_window        = 1;   {frame window}
  left_margin         = 1;   {beginning column of each line.}
  top_row             = 1;   {beginning row of each page starts here.}
  exit_window         = 3;   {exit window}
  editor_window       = 5;   {editor window}
  warning_window      = 6;   {warning window}
  hlp_win             = 8;   {help window}
  Erase               = #8;  {erase}
  main_screen         = 21;  {main screen}
  space               = #32; {space}
  f1                  = 59;  {keyboard function f1}
  f2                  = 60;  {keyboard function f2}
  f3                  = 61;  {keyboard function f3}
  f4                  = 62;  {keyboard function f4}
  f5                  = 63;  {keyboard function f5}
  f6                  = 64;  {keyboard function f6}
  f7                  = 65;  {keyboard function f7}
  f8                  = 66;  {keyboard function f8}
  f9                  = 67;  {keyboard function f9}
  f10                 = 68;  {keyboard function f10}

  {FUNCTION KEYS, ARROWS, ETC.  USE AN ASCII ESCAPE SEQUENCE}

  bksp                =   8; {backspace; rub out char to left of cursor}
  tab                 =   9; {tab right to the first blank space.}
  enter               =  13; {new line. inserts blank line if INS pressed.}
  s_tab               =  15; {shift/tab key; tab left to 1ST blank space.}
  escape              =  27; {escape key}
  home                =  71; {home key; go to top of file, or first page}
  up                  =  72; {up arrow key}
  pgup                =  73; {goto previous page}
  left                =  75; {left arrow key}
  right               =  77; {right arrow key}
  endkey              =  79; {go to page with last line of text on it. }
  down               =  80; {down arrow key}
  pgdn               =  81; {goto next page}
  ins                =  82; {insert mode used for character and enter.}
  del                =  83; {delete the cursor line.}
  s_f1               =  84; {shift key/F1 - help screen for text editor. }
  s_f2               =  85; {shift key/F2 - save text in file & exit editor.}
  alt_f9             = 112; {alt key/f9 - exit send msg screen to main menu.}
  ctrl_left          = 115; {ctrl/leftarrow; goto beginning of line.}
  ctrl_right         = 116; {ctrl/rightarrow; goto end of line.}
  ctrl_end           = 117; {ctrl/endkey; delete from cursor to eol}
  ctrl_bksp          = 127; {control backspace key}

TYPE
  str2       = STRING[2];    {string for user id}
  u_exten    = STRING[2];    {user extension}
  string12   = STRING[12];   {used for warning message constants}
  txstr      = STRING[80];   {string for a line of text}
  string160  = STRING[160];  {string for wrap around text}
  screenloc  = RECORD
                 character: CHAR;
                 attribute: Byte;
               END;
  videoram   = ARRAY[1..2000] OF screenloc;

{*********************** RECORD FOR LINKED LIST NODE *************************}
  nodeptr = ^node;
  node    = RECORD
              txt: txstr;
              next: nodeptr;
              prior: nodeptr;
              newln: INTEGER;
            END;

{*********************** RECORD FOR LINKED LIST HEADER ***********************}
  list = ^head;
  head = RECORD
           LENGTH: INTEGER;
           first:  nodeptr;
           last:   nodeptr;
         END;
  file_len_66 = STRING[66];
  textstr     = STRING[80];

{*********************** RECORD FOR INPUT WINDOW NODE ************************}
  winnode  = ^wins;
  wins     = RECORD
               ex1    : INTEGER;
               ex2    : INTEGER;
               fldlen : INTEGER;
               lyne   : INTEGER;
               STR    : textstr;
               next   : winnode;
               prior  : winnode;
             END;
```

```
{*********************** RECORD FOR INPUT WINDOW LIST HEADER *******************}
  winlsthdr = ^wnhead;
  wnhead    = RECORD
               box     : ARRAY [1..4] OF INTEGER;
               LENGTH : INTEGER;
               bgclr  : INTEGER;
               fgclr  : INTEGER;
               menchr : winlsthdr;
               savitem: winnode;
               first  : winnode;
               last   : winnode;
             END;



{****************************************************************************
*                        VARIABLES FOR THE HEART OF THIS PROGRAM            *
****************************************************************************}
VAR

  key,                                {key value for the file}
  ch,                                 {trapped character from keyboard}
  bell              : CHAR;           {used to ring bell}
  done,                               {flag used in bldmsg}
  wasfound,                           {sets to true if name is in idfile.dat}
  first_time,                         {indicates to getime, been thru once}
  canceled_message,                   {indicates msg cancellation}
  insertmd,                           {indicates insert mode is on}
  is_there_text     : BOOLEAN;        {indicates if the message is blank}
  chour,                              {character hour}
  cmin              : STRING[2];      {character minute}
  ctime             : STRING[5];      {character time e.g. 10:13}
  tmp,                                {temporary holder of the user id}
  to_name,                            {user name that msg is being sent to}
  from_name         : STRING[20];     {user name that send the msg}
  from_and_from_name,                 {contains FROM:  and sender's user name}
  to_and_to_name    : STRING[27];     {contains TO:  and receiver's user name}
  menu_name         : STRING[66];     {main menu name}
  blanks            : STRING[80];     {blank line}
  var2,                               {holds value returned form use_menu}
  vchoice,                            {holds value of function key choice}
  n_cur_pos,                          {current position ptr for new mail msgs}
  n_add_pos,                          {cur pos ptr for next avail pos new msg}
  o_cur_pos,                          {position ptr for old mail msgs}
  o_add_pos,                          {pos ptr for next avail pos for new msg}
  choice,                             {selection choice of function key}
  recv_id,                            {receiver's user name}
  savex,                              {used to save cursor position x}
  savey,                              {used to save cursor position y}
  colset,                             {column number of upper left position}
  colset1,                            {column number of lower right position}
  rowset,                             {row number of upper left position}
```

```
rowset1,                          {row number of lower left position}
i,                                {loop control variable}
j,                                {loop control variable}
start,                            {start position}
count,                            {counter}
insert_column,                    {insert column}
insert_row,                       {insert row}
max_lines,                        {maximum number of lines}
bottom_row,                       {last row number}
column,                           {column number}
row,                              {row number}
temp_column,                      {temporary holder of column number}
colour,                           {background color}
hlpcolor,                         {help color}
string_length,                    {string length}
right_margin,                     {right margin}
clf,                              {left column}
cnt_flag,                         {count flag}
node_ln,                          {number of node in linked list}
color,                            {color}
x1,                               {left column of window}
y1,                               {top line of window}
x2,                               {right column of window}
y2,                               {bottom line of window}
back,                             {background color of window}
fore,                             {color of text in window}
fback,                            {bgd color of border}
ffore,                            {color of line in border}
bor,                              {# of border lines}
outpt,                            {1=rewrite and 2=append}
hour,                             {hour value}
hour1,                            {hour value from user terminal}
hour2,                            {hour value from the system}
min,                              {minute value}
min1,                             {minute value from user terminal}
min2               : INTEGER;     {minute value from the system}
user_file,                        {designates the user file}
getfile,                          {in getime for mail message display}
sub_fil_str,                      {user file string}
fil,                              {designates file string}
edt_file           : str_66;      {contains the sender's message}
erase_fil,                        {file str used to erase unneeded msgs}
quitfile,                         {file string for cancelled messages}
old_fil,                          {file string for the old messages}
timefile           : TEXT;        {file string containing the base time}
wrapper            : string160;   {string for wrap around text}
blankstr           : txstr;       {blank string}
var1               : winlsthdr;   {holds value returned from read_menu_file}
unum_ext,                         {user number extension}
fnum_ext           : u_exten;     {file number extension}
Lst                : list;        {linked list}
curnd              : nodeptr;     {current node}
```

```
{*******************************************************************
*  GET_FROM_NAME;  AUTOMATICALLY GETS SENDER'S NAME FROM IDFILE.DAT *
********************************************************************}
PROCEDURE get_from_name;

VAR
   t_name    : STRING[20];
   t_id      : INTEGER;

BEGIN
   Assign(idlist,idfile_dat);
   lock;
   RESET(idlist);
   ctr := send_id - 1;
   seek(idlist,ctr);
   read(idlist,an_entry);
   with an_entry do
     begin
       from_name := name;
     end;
   CLOSE(idlist);
   unlock;
END;


{***************************************************************************
*  GET_TIME;  GETS THE HOURS AND MINUTES FROM THE USER TERMINAL             *
***************************************************************************}
PROCEDURE get_time(VAR hour, min : INTEGER);
TYPE
  regpack = RECORD
              ax,bx,cx,dx,bp,si,di,ds,es,flags: INTEGER;
            END;
VAR
  regs : regpack;

BEGIN
  WITH regs DO
  BEGIN
    ax   := $2c00;
    MSDos(regs);
    hour := Hi(cx);
    min  := Lo(cx);
  END;
END;


{*******************************************************************
*  HOUR_CALC;  USED IN CALCULATING THE HOUR IF THERE IS OVERLAP     *
*              BETWEEN A SESSION AND THE 24TH HOUR                  *
********************************************************************}
PROCEDURE hour_calc;
 BEGIN
  IF(hour2 >= 1) THEN
```

```
    IF(min1+min2 >= 60) THEN
      hour:=hour2+1-(24-hour1)
    ELSE
      hour:=hour2-(24-hour1)
  ELSE
    hour:=1;
  IF (hour=0) THEN hour:=24;
END;


{****************************************************************************
*  TIMESTAMP;  CALCULATES TIME WHICH A MESSAGE IS SENT                      *
****************************************************************************}
PROCEDURE Timestamp;
 BEGIN
  get_time(hour1, min1);           {get the time from the terminal}
  Assign(timefile, time_file);
  RESET(timefile);
  READLN(timefile, hour2, min2);   {get base time established when}
  CLOSE(timefile);                 {EBS was initiated.           }
  IF(min1+min2 >= 60) THEN
    BEGIN
      min:=min1+min2-60;
      IF(hour1+hour2+1 > 23) THEN hour_calc ELSE hour:=hour1+hour2+1;
    END
  ELSE
    BEGIN
      min:=min1+min2;
      IF(hour1+hour2 > 23) THEN hour_calc ELSE hour:=hour1+hour2;
    END;
  STR(hour, chour);
  STR(min, cmin);
  IF(hour < 10) THEN
    BEGIN
      chour[2]:=chour[1];
      chour[1]:='0';
    END;
  IF(min < 10) THEN
    BEGIN
      cmin[2]:=cmin[1];
      cmin[1]:='0';
    END;
  ctime:='     ';
  ctime:=chour+':'+cmin;
END;


{****************************************************************************
* SEND;      INCLUDE FILE THAT PREPARES MESSAGE FOR SENDING                 *
* MENU;      INCLUDE FILE THAT BRINGS UP THE MAIN EMAIL MENU                *
* STRMENU;   INCLUDE FILE USED TO DISPLAY USERNAME LIST IN BLDMSG.INC       *
* BLDMSG;    INCLUDE FILE THAT ALLOWS THE USER TO ENTER HIS MESSAGEG        *
*****************************************************************************}
{$I B:SEND.INC}
```

```
{$I B:MENU.INC}
{$I B:STRMENU.INC}
{$I B:BLDMSG.INC}


{*****************************************************************************
*     GET_RECEIVER;  GETS THE RECEIVER'S ID# AND NAME FROM IDFILE_DAT       *
*****************************************************************************}
PROCEDURE get_receiver;


BEGIN
  wasfound := FALSE;
  Assign(idlist,idfile_dat);
  lock;
  RESET(idlist);
  ctr := 1;
  while (not wasfound) and (ctr in [1..nodes]) do
   begin
     seek(idlist,ctr-1);
     read(idlist,an_entry);
     with an_entry do
       begin
        IF (name = to_name) THEN
          begin
            wasfound := TRUE;
            recv_id := id;
          end;
       end;
     ctr := ctr + 1;
   end;
  CLOSE(idlist);
  unlock;
END;       { end procedure get_receiver }

{*****************************************************************************
*  PUT_DIR; TAKES THE SENDER AND RECEIVER ID'S AND LOCATES AN OPEN POSITION*
*           IN THE RECEIVER'S DIRECTORY AND COPY 'EDT.FIL' FROM THE SENDER *
*           DIRECTORY.                                                     *
*****************************************************************************}
PROCEDURE put_dir;
CONST
  hdr        = '           TIME MESSAGE SENT: ';

TYPE
  filename   = STRING[66];         {file name}

VAR
  cur_pos    : INTEGER;             {current msg pos w/i the file directory}
  u_ext,                            {user string number}
  f_ext      : STRING [2];          {file number concat onto file string}
  valid_line : STRING [80];         {sender file valid text line storage}
  open_pos   : BOOLEAN;             {signals open position w/i directory}
  sendfile,                         {msg name to be sent in sender's dir}
  recvfile   : filename;          {name of message in receiver's dir}

BEGIN
  sendfile := user_filetable[send_id] +  send_ext;
  open_pos := FALSE;
  cur_pos  := 0;
  STR (recv_id, u_ext);
  WHILE NOT open_pos DO
    BEGIN
      cur_pos := cur_pos + 1;
      STR (cur_pos, f_ext);
      recvfile := user_filetable[recv_id] + user_ext + u_ext +
                  new_ext + f_ext;
      IF NOT exist(recvfile)  THEN
        BEGIN
          open_pos := TRUE;
          Assign(outfile, recvfile);
          REWRITE(outfile);
        END;
    END;
  Assign(infile, sendfile);
  RESET(infile);
  IF log_on AND (not for_ward) AND ((NOT broadcast) OR (broadcast AND (bt = 1))) THEN
    BEGIN
      append(log_file);
      WHILE NOT EOF(infile) DO
        BEGIN
          READLN(infile,valid_line);
          writeln(outfile,valid_line);
          writeln(log_file,valid_line);
        END;
      writeln(log_file);
      writeln(log_file,'** End of Message **');
      writeln(log_file,'----------------------------------------------------------');
      writeln(log_file);
      CLOSE(log_file);
    END
  ELSE
    BEGIN
      if (for_ward) then
        begin
          timestamp;  {get the time the message is sent}
          writeln(outfile,from_and_from_name,hdr,chour[1],chour[2],':',cmin[1],cmin[2]);
          writeln(outfile,to_and_to_name);
          writeln(outfile);
        end;
      WHILE NOT EOF(infile)  DO
        BEGIN
          READLN(infile, valid_line);
          writeln(outfile, valid_line);
        END;
    END;
  CLOSE(infile);
  IF NOT broadcast THEN  Erase(infile);
  writeln(outfile);
  if (not for_ward) then
     writeln(outfile,'** End of Message **');
  CLOSE(outfile);
END;                  { end procedure Put_dir }


{*****************************************************************************
*  GET_MSG;  GETS THE MESSAGE ENTERED BY THE SENDER                          *
*****************************************************************************}
PROCEDURE get_msg;

{$I B:GET.INC}

BEGIN
  first_time:=TRUE;
  STR (send_id, unum_ext);
  sub_fil_str:= usr_path + unum_ext;
  getfile:= user_ext + dum_ext;
  x1:=2;
  y1:=6;
  x2:=79;
  y2:=18;
  back:=blue;
  fore:=yellow;
  fback:=cyan;
  ffore:=yellow;
  edt_file:=getfile;
  bor:=2;
  window_reader;
END;             { end procedure Get_msg }


{*********************** DRIVER:  MAIN PROGRAM *****************************}
BEGIN
  WHILE KeyPressed DO READ(Kbd,keychr);     {clear any waiting keys}
  from_name := '                      ';
  menu_name := m_menu_dat;
  done:=FALSE;
  bell := CHR(7);
  reply := false;
  for_ward := false;
  REPEAT
   if (not reply) and (not for_ward) then
     begin
       putx(1,1,pic);
       broadcast := FALSE;
       cursoron(FALSE);
       MARK(heaptop);
       var1 := read_menu_file(menu_name);
       var2 := use_menu(var1,2,blue,white,FALSE);       {display main menu}
       if (var2 = 1) or (var2 = 2) then
         begin
           get_from_name;                              {get sender's username}
           if (from_name = twyblks) then
             var2 := 3;
         end;
       cursoron(TRUE);
       choice := var2;
       RELEASE(heaptop);
       TextBackGround(black);
       ClrScr;
     end
   else
     begin
       textbackground(black);
       clrscr;
       choice := 1;
     end;

{**************************************************************************
* CASE 1;  SEND MESSAGE                                                   *
**************************************************************************}
    CASE choice OF
      1:  BEGIN
            bt := 1;
            MARK(heaptop);
            skip := false;
            bld_msg;                           {build the message}
            if (skip) and (for_ward) then
              erase(infile);
            if (not skip) then                 {if escape key not pressed then continue}
              begin
                reply := false;
                Assign(listid, idfile_dat);
                Window(1,1,80,25);
                putx(1,1,pic);
                IF NOT (canceled_message) THEN
                  BEGIN
                    mkwin(28,11,43,15,black+blink,green,1);
                    cursoron(FALSE);
                    writeln;
                    writeln('***WORKING***');
                    IF broadcast THEN
                      BEGIN
                        lock;
                        RESET(listid);
                        ctr := 1;
                        while ctr in [1..nodes] do
                          begin
                            seek(listid,ctr-1);
                            read(listid,an_entry);
                            with an_entry do
                              begin
                                recv_id := id;
                                if (name <> blanks20) then
                                  begin
                                    put_dir;           {put mail in user's dir.}
                                    bt := bt+1;         {set flag, used by put_dir}
                                  end;
                              end;
                            ctr := ctr + 1;
                          end;
                        Erase(infile);
                        CLOSE(listid);
                        unlock;
                      END
                    ELSE
                      begin
                        get_receiver;   { get the receiver's username }
                        if wasfound then
                          put_dir       { put mail in user's dir. }
                        else
                          begin
                           assign(infile,in_fil_str);
                           erase(infile);
                          end;
                      end;
                    rewin;
                  END;                  { end if not canceled_message }
                cursoron(TRUE);
              end;                      { end if not skip }
            canceled_message := FALSE;
            RELEASE(heaptop);
            for_ward := false;
        END;

{**************************************************************************
* CASE 2; READ MESSAGE                                                    *
***************************************************************************}
      2:  BEGIN
            get_msg;
            Window(1,1,80,25);
          END;

{**************************************************************************
* CASE 3; RETURN TO PREVIOUS SCREEN                                       *
***************************************************************************}
      3:  done:=TRUE;
    END;
  UNTIL done;
END;                            { exit the driver}


{----------------------------------------------------------------------------}
{                   THE ABOVE ARE THE USER INCLUDE ROUTINES                  }
{----------------------------------------------------------------------------}

{----------------------------------------------------------------------------}
{                    P R O C E S S   I N T E R R U P T                       }
{----------------------------------------------------------------------------}
{  PURPOSE:
      The following procedures  displace  standard interrupts.

      Do not put Variables or Constants in this Procedure. It will
      cause registers to be clobbered during the Interrupt routine
      when Turbo attempts to allocate  storage for local variables
      or parameters.}

PROCEDURE stay_int16;       {Keyboard Interrupt 16 Service Routine}

      {If anything but "Our_HotKey" is pressed,  the key is  passed
      to the standard  keyboard service routine.  B_U_T,  when Our
      HotKey is recognized, a hotkey bit is set.}

BEGIN
{$I b:STAYI16.410}
END; {STAY_INT16}

PROCEDURE stay_int13;       {BIOS Disk interrupt  Routine}
BEGIN                       {Sets a flag while disk is active}
{$I b:STAYI13.410}
END; {STAY_INT13}

PROCEDURE stay_int21;       {DOS interrupt 21 Service Routine}
BEGIN                       {Sets a flag while INT 21 is active}
{$I b:STAYI21.410}
END; {STAY_INT21}

PROCEDURE stay_int8;        {Timer Interrupt 8 Service Routine}
                            {Activates Stayres during pgm execution}
BEGIN                       {when safe to do so.}
{ add the following include for the Clock Demonstration by Neil Rubenking }
{$I b:STAYI8.420}
END;{Stay_Int8}

PROCEDURE stay_int28;       {Idle Interrupt 28 Service Routine}
BEGIN                       {Invokes Stayres from the DOS prompt}
{$I b:STAYI28.410}
END;{Stay_Int28}            {continue}

PROCEDURE staysave;         {Prolog to Resident Turbo Code}
BEGIN
{$I b:STAYSAVE.420}
  getdta(saveddta[1],saveddta[2]);          {Save callers DTA address}
  getpsp(savedpsp);                         {Save callers PSP Segment}
  setpsp(ourpsp);                           {Set our PSP Segment}
  setdta(ourdta[1],ourdta[2]);              {Set our DTA address}
  newctlc[2] := CSeg;
  newctlc[1] := Ofs(iret);
  getctlc(savedctlc); setctlc(newctlc);     {Get/Save the users Ctrl-C vector}
  int24on;                                  {Trap Dos Critical Errors}

        {-------------------------------------------------------}
        {              INVOKE USER PROCEDURE HERE               }
        {-------------------------------------------------------}
```

```
     BEGIN
      xp := wherex;           {get the main x cursor position}
      yp := wherey;           {get the main y cursor position}
      pic := getx(1,1,80,25);{take a picture of the screen}
      keychr := #0;           {clear any residual}
      driver;                 {call main routine}
      xfree := true;          {set flag to release image in putx}
      putx(1,1,pic);          {put the user screen back}
      xfree := false;
      gotoxy(xp,yp)           {put the cursor back where it was}
     END;

    {-------------------------------------------------------------}
    {              END USER PROCEDURE HERE                         }
    {-------------------------------------------------------------}

 setpsp(savedpsp);                 { Restore Callers PSP Segment}
 setdta(saveddta[1],saveddta[2]);  { Restore the users DTA}
 setctlc(savedctlc);               { Restore the users Ctrl-C Vector}
 int24off;                         { Remove Our Critical Error routine}


{-------------------------------------------------------------------------------}
{                BEGINNING OF THE STAYRSTR ROUTINE                              }
{-------------------------------------------------------------------------------}
{$I b:STAYRSTR.420}

     {-------------------------------------------------------}
     {       END OF THE STAYRSTR ROUTINE                      }
     {-------------------------------------------------------}

END;
{-------------------------------------------------------------------------------}
{                           M A I N                                             }
{-------------------------------------------------------------------------------}
     { The main program installs the new interrupt routine }
     { and makes it permanently resident as the keyboard   }
     { interrupt.  The old keyboard interrupt Vector is    }
     { stored in Variables , so they can be used in Far    }
     { Calls.                                              }
     {                                                     }

     { The following dos calls are used:                   }
     { Function 25 - Install interrupt address             }
     {               input al = int number,                }
     {               ds:dx = address to install            }
     { Function 35 - get interrupt address                 }
     {               input al = int number                 }
     {               output es:bx = address in interrupt   }
     { Function 31 - terminate and stay resident           }
     {               input dx = size of resident program   }
     {               obtained from the memory              }
     {               allocation block at [Cs:0 - $10 + 3]  }
```

```
          { Function 49 - Free Allocated Memory                  }
          {               input Es = Block Segment to free       }
          {------------------------------------------------------}

BEGIN
  ourdseg:= DSeg;               {Save the Data Segment Address for Interrupts}
  oursseg:= SSeg;               {Save our Stack Segment for Interrupts}
  getpsp(ourpsp);               {Local PSP Segment}
  getdta(ourdta[1],ourdta[2]); {Record our DTA address}
  userprogram:=Ofs(staysave);   {Set target of call instruction}
  regs.ax := $3000 ;            {Obtain the DOS Version number}
  Intr(dosi21,regs);
  dosversion := halfregs.al;    { 0=1+, 2=2.0+, 3=3.0+ }
                                {Obtain the DOS Indos status location}
  regs.ax := $3400;
  Intr(dosi21,regs);
  dosstat1.ip  := regs.bx;
  dosstat1.cs  := regs.es;
  dosstat2.cs  := regs.es;
  dossseg      := regs.es;
  bytecount := 0;               {Search for CMP [critical flag],00 instruct.}
    WHILE (bytecount < $2000)   {then Mov SP,stackaddr instruction }
      AND (MemW[dosstat2.cs:bytecount] <> $3e80)
      DO bytecount := SUCC(bytecount);
      IF bytecount = $2000 THEN      { Couldn't find critical flag addr }
        BEGIN
          writeln('StayRes incompatiblity with Operating System');
          writeln('StayRes will not install correctly..Halting');
          HALT;
        END;

      { Search for the DOS Critical Status Byte address.       }
      { Bytecount contains offset from DosStat1.CS of the      }
      {        CMP [critical flag],00                          }
      {        JNZ ....                                        }
      {        Mov SP,indos stack address                      }

      IF Mem[dosstat2.cs:bytecount+7] = $bc THEN   {MOV SP,xxxx}
        BEGIN
          dosstat2.ip := MemW[dosstat2.cs:bytecount+2];
          dossptr     := MemW[dosstat2.cs:bytecount+8]; {INDOS Stack address}
        END
      ELSE
        BEGIN
           writeln('Cannot Find Dos Critical byte...Halting');
           HALT;
        END;
      InLine($fa);                        {Disable interrupts}


      { Setup Our Interrupt Service Routines }

      setup_interrupt(biosi16, bios_int16, Ofs(stay_int16)); {keyboard}
      setup_interrupt(biosi8, bios_int8, Ofs(stay_int8));    {timer}
```

```
     setup_interrupt(biosi13, bios_int13, Ofs(stay_int13)); {disk}
     setup_interrupt(dosi21, dos_int21, Ofs(stay_int21));   {DOSfunction}
     setup_interrupt(dosi28, dos_int28, Ofs(stay_int28));   {DOS idle}
     InLine($fb);                          {Re-enable interrupts}

{****************************************************************************}
{----------------------------------------------------------------------------}
{                     INITIALIZE YOUR PROGRAM HERE                           }
{----------------------------------------------------------------------------}
{****************************************************************************}
     { Initialize Program Here since we will not get control again
       until "Our_HotKey" is entered from the Keyboard.             }

     log_on:=FALSE;
     Assign(user,usr_id_str);
     RESET(user);
     READLN(user,send_id);         { get user's ID number from USER.ID }
     CLOSE(user);
     TextColor(white);
     writeln;
     writeln;
     writeln('*** EMAIL SYSTEM IS NOW RESIDENT. ***');
     writeln('*** PRESS SHIFT/F7 TO ENTER EMAIL ***');
     writeln;
     writeln;
     done   :=FALSE;
     xfree  := FALSE;
     msgint :=1;                    {look for message number 1}
     IF exist(log_str) THEN
       BEGIN
         log_on:=TRUE;              {if email.log exists, then user has re-}
         Assign(log_file,log_str);  {quested a log, get it ready and set}
       END;                         {flag so put_dir will know}
     terminate := FALSE;            {clear the program exit flag}
                                    {don't want to take email out of mem}
                                    {if set to true this pgm will end}

    {------------------------------------------------------------}
    {              END OF INITALIZE PROGRAM CODE                 }
    {------------------------------------------------------------}

    {Now terminate and stay resident.  The following Call utilizes the
    DOS Terminate & Stay Resident function.  We get the amount of
    memory by fetching the memory allocation paragraphs from the Memory
    Control Block.  This was set by Turbo initialization during Int
    21/function 4A (shrink block), calculated from the mInimum and mAximum
    options menu. The MCB sits one paragraph above the PSP.}

                                        { Pass return code of zero    }
    regs.ax := $3100 ;                  { Terminate and Stay Resident }
    regs.dx := MemW [CSeg-1:0003]+1 ;   { Prog_Size from Allocation Blk}
    Intr (dosi21,regs);
END.    { END OF RESIDENCY CODE }
```

APPENDIX B

PROGRAM LISTING

of

ECHECK.PAS

```
{###############################################################################
#                                                                             #
# ECHECK.PAS          GARY MCALUM              August 1987                    #
#                                                                             #
#                                                                             #
# PROGRAM DESCRIPTION:                                                        #
#                                                                             #
# ECHECK is a memory-resident program which is used in conjunction with      #
# another program called EMAIL.  Together they form the network messaging    #
# system for PLEXSYS. ECHECK scans each user's directory, every 15 seconds,#
# to see if there is any new mail.  If so, it notifies the user with an      #
# audible and visible signal.  In addition, this program performs the        #
# function of checking for screen messages, also called "notes".  These      #
# notes are created and sent from the DOS prompt by a program called SEND. #
# Once the note has been placed in the user's directory with the extension #
# .MES then this program will notify the receiver by and audible signal      #
# and by displaying the actual note on the screen.  The user has to press   #
# C to remove the message.                                                    #
#                                                                             #
# INPUT/OUTPUT FILES:                                                         #
#                                                                             #
# (1)  USER.ID      : contains the unique user number in ASCII (1-16)        #
# (2)  USERX.MES    : X = user #, .MES file is on-screen note                #
# (3)  USERX.N#     : X = user #, # = 1,2,3,...,  new mail message           #
#                                                                             #
# PROGRAM REQUIREMENTS:                                                       #
#                                                                             #
# Memory required is approximately 30k bytes of RAM.  Any type of disk       #
# drive is compatible since the program is loaded directly in memory and    #
# does not access the disk containing the source program afterwards.        #
# ECHECK was developed and tested using MS/DOS Version 3.1 and TURBO Pascal#
# Version 3.0.  This program should be assembled as a COM file with mini-   #
# mum dynamic memory set to 0275 and max. dynamic memory set to 0275. This #
# ensures enough memory is allocated to the heap to avoid collisions while #
# limiting the amount of memory reserved for the heap.                      #
#                                                                             #
###############################################################################}

{$R+}
{$C-}

PROGRAM echeck;

{####################### CONSTANTS USED IN THE SHELL ########################}

CONST
  maxwin     = 10;      {Max # windows open at 1 time, used in window.inc}
  esc        = #27;     {character equivalent of Escape Key}
  alt_f9     = #112;    {Alt Function 9 Scan code        }
  ctrl_f9    = #229;    {Ctl-F9 (#102+127)  Key  code     }
  quit_key   = ctrl_f9; {Quit and Release Memory}
  alt        = 08;      {Shift bits at 40:17 }
  ctrl       = 04;      {the code for the control key}
  left_shift = 02;      {used to shift left}
  rght_shift = 01;      {used to shift right}
  biosi8     = 8;       {Bios Timer interrupt}
  biosi16    = $16;     {Bios Keyboard interrupt}
  biosi13    = $13;     {Bios Disk interrupt}
  dosi21     = $21;     {DOS service router interrupt}
  dosi28     = $28;     {DOS Idle interrupt}

{$$$$$$$$$$$$$$$$$$$$$$ TYPE DECLARATIONS USED IN THE SHELL $$$$$$$$$$$$$$$$$}

TYPE
  regtype       = RECORD
                    ax,bx,cx,dx,bp,si,di,ds,es,flags:INTEGER
                  END;
  halfregtype   = RECORD
                    al,ah,bl,bh,cl,ch,dl,dh:Byte
                  END;
  filename_type = STRING[64];
  str_66        = STRING[66];
  vector        = RECORD                      {Interrupt Vector type}
                    ip,cs :INTEGER ;
                  END ;
  rayptr        = ^bigray;
  bigray        = RECORD
                    nc,nl:INTEGER;
                    pixray: ARRAY[1..4000] OF Byte;
                  END;

{$$$$$$$$$$$$$$$$$$$$$$ TYPED CONSTANTS USED IN THE SHELL $$$$$$$$$$$$$$$$$$$}

CONST
  our_hotkey   : Byte = 90;                   {scan code for SHIFT-F7}

{$$$$$$$$$$$$$$$$$$$$$$ SCAN CODE CAN BE CHANGED TO MAKE $$$$$$$$$$$$$$$$$$$$}
{$$$$$$$$$$$$$$$$$$$$$$ ANOTHER KEY ACTIVE AS THE HOST KEY. $$$$$$$$$$$$$$$$$}

     { This table marks those INT 21 functions which must be passed     }
     { without modification. They either never return, fetch parameters }
     { from the stack, or may be interrupted by a TSR.                  }

     functab    : ARRAY[0..$6f] OF Byte =
                   (1,1,1,1, 1,1,1,1, 1,1,1,1, 1,0,0,0,  {0-C}
                    0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0,
                    0,0,0,0, 0,0,1,0, 0,0,0,0, 0,0,0,1,  {26,2F}
                    0,1,1,1, 1,1,0,0, 0,0,0,0, 0,0,0,0,  {31-35}
                    0,0,0,0, 0,0,0,0, 1,1,1,1, 1,1,0,0,  {48-4D}
                    1,1,1,1, 0,1,0,0, 1,0,0,0, 0,1,1,1,  {50-53,55,58,5D-5F}
                    1,1,1,1, 1,1,1,1, 1,1,1,1, 1,1,1,1); {60-62}

     intr_flags  : Byte = 0;          {Active interrupts flags}
       int13_on  =  04;               {Disk  interrupt is active}
       int21_on  =  08;               {DOS Service router is active}
     status      : Byte = 0;          {Status of current TSR activity}
       hotkey_on =  01;               {Received the HotKey}
```

```
      inuse      =  02;              {TSR is active}
      foxs       = $ff;              {workaround for inline hex FF}
    dosversion  : Byte = 0;          {Current Version of DOS}
    waitcount   : Byte = 0;          {Wait to activate count}
    userprogram :INTEGER = 0;        {Offset to Users Program Code}
    ourdseg     :INTEGER = 0;        {Turbo Data Segment Value  }
    oursseg     :INTEGER = 0;        {Turbo Stack Segment Value   }
    dosdseg     :INTEGER = 0;        {Dos Datasegment value      }
    dossseg     :INTEGER = 0;        {Dos Stack Segment Value    }
    dossptr     :INTEGER = 0;        {Dos Stack pointer value    }
    dosssiz     :INTEGER = 0;        {Dos Stack size in words }
    usrdseg     :INTEGER = 0;        {Interrupted Datasegment value       }
    usrsseg     :INTEGER = 0;        {Interrupted Stack Segment Value    }
    usrsptr     :INTEGER = 0;        {Interrupted Stack pointer value    }
    ourpsp      :INTEGER = 0;


  { The following  constants *MUST* remain in the IP:CS order.   }
  { StaySave uses them as  JMP targets                           }

    bios_int8   :vector = (ip:0;cs:0);  {BIOS Timer Interrupt Vector }
    bios_int16  :vector = (ip:0;cs:0);  {BIOS Keyboard Interrupt Vector }
    bios_int13  :vector = (ip:0;cs:0);  {BIOS Disk Interrupt Vector     }
    dos_int21   :vector = (ip:0;cs:0);  {DOS Sevice Interrupt Vector}
    dos_int28   :vector = (ip:0;cs:0);  {DOS idle Service interrupt Vector}
    dosstat1    :vector = (ip:0;cs:0);  {Pointer to INDOS byte}
    dosstat2    :vector = (ip:0;cs:0);  {Pointer to CRITICAL byte}

    version     :STRING[4] = '4.15';    { Current Version number }

    {NEEDED FOR SETTIME}

    timer_message :STRING[80] = '';
    timer_on    = 4;                      {flag set when timer is on}
    from_timer = 8;                       {flag set when timer expires}
    timer_hi    :INTEGER = 0;
    timer_lo    :INTEGER = 0;
    new_ext     :STRING[2] = '.N';        {used to designate new mail msgs}
    note_ext    :STRING[4]  = '.mes';     {contains note message}
    usr_id_str :str_66 = 'D:\MAIL\USER.ID';{contains user's id number}
    usr_path    :str_66 = 'D:\MAIL\USER';   {init path strg w/i ea user dir}


{************ CHANGE TIMER_TIME TO THE VALUE (IN SECONDS) THAT ************'$**}

    timer_time     = 15;               {in secs, time new mail msg shown}

{************ YOU WANT THE PROGRAM TO CHECK FOR NEW MESSAGES ***************}



{************************ VARIABLES **********************************************}

VAR
```

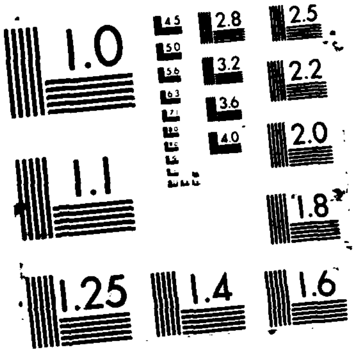
```
  regs       : regtype;
  halfregs   : halfregtype Absolute regs;
  keychr     : CHAR ;
  bytecount  : INTEGER;
  savedpsp   : INTEGER;           { Program Segment Prefix pointers }
  error      : INTEGER;           { I/O results }
  good       : BOOLEAN;           { I/O results switch }
  ourdta   :ARRAY [1..2] OF INTEGER; {Local DTA pointer}
  saveddta :ARRAY [1..2] OF INTEGER; {Interrupted DTA pointer}

  {NEEDED FOR MAIL PROGRAM}

  tics               :REAL;       {used to set timer}
  xfree,                          {flag used to signal dispose(ray)}
  done               :BOOLEAN;    {flag used in bldmsg}
  msgnum             :STRING[2];  {message number of the mail}
  usrno              :STRING[2];  {user number}
  ctime              :STRING[5];  {character representation of the time}
  heaptop            :^INTEGER;   {used to mark top of heap}
  hiclock            :INTEGER Absolute $40 :$6e;
  loclock            :INTEGER Absolute $40 :$6c;
  num,                            {number of actual messages, Chk_num}
  msgint,                         {message number in integer format}
  tpos,                            {cursor position for note window}
  send_id,                        {sender's user number}
  xp,                             {used in mail_man for xcursor position}
  yp                 :INTEGER;    {used in mail_man for ycursor position}
  inp_char           :CHAR;
  a_line             :STRING[65];
  note,
  user               :TEXT;       {user idfile string}
  newnote,                        {contains the note message}
  numfile,                        {temp. work file for Chk_num}
  newfile            :str_66;     {file string name, locates new mail files}
  pic                :rayptr;     {ptr to orig user screen}

{---------------------------------------------------------------------------}
{                 W I N D O W   R O U T I N E                               }
{---------------------------------------------------------------------------}
{$I b:STAYWNDO.341}

{###########################################################################}
{---------------------------------------------------------------------------}
{               THE FOLLOWING ARE THE USER INCLUDE ROUTINES                 }
{---------------------------------------------------------------------------}
{###########################################################################}
{$I b:STAYSUBS.420}

{---------------------------------------------------------------------------}
{                 PROCEDURE SETTIME NEEDED TO INITIALIZE                    }
{---------------------------------------------------------------------------}
     {-------------------------------------------------------------}
     {                DOUBLE TO REAL NUMBER CONVERSION             }
     {-------------------------------------------------------------}
     FUNCTION double_to_real(i,j : INTEGER):REAL;
     VAR temp : REAL;
     BEGIN
       temp := i;
       IF temp < 0 THEN temp := temp + 65536.0;
       temp := temp * 65536.0;
       IF j < 0 THEN temp := temp + 65536.0 + j ELSE temp := temp + j;
       double_to_real := temp;
     END;

     {--------------------------------------------------------------}
     {               REAL TO DOUBLE NUMBER CONVERSION               }
     {--------------------------------------------------------------}

     PROCEDURE real_to_double(r : REAL; VAR i, j : INTEGER);
     VAR it, jt : REAL;
     BEGIN
       it := INT(r/65536.0);
       jt := r ~ it*65536.0;
       IF it > MAXINT THEN i := TRUNC(it - 65536.0) ELSE i := TRUNC(it);
       IF jt > MAXINT THEN j := TRUNC(jt - 65536.0) ELSE j := TRUNC(jt);
     END;

     {--------------------------------------------------------------}
     {               SET TIME    TURN TIMER ON                      }
     {--------------------------------------------------------------}

     PROCEDURE set_timer(the_time : INTEGER);
     BEGIN
       tics := double_to_real(hiclock, loclock);
       tics := tics + the_time*18.206481934;
       real_to_double(tics, timer_hi, timer_lo);
       status := status OR timer_on;
     END;

{****************************************************************************
*  EXIST;  RETURNS TRUE IF this_file EXISTS IN THE DIRECTORY               *
****************************************************************************}
FUNCTION exist(this_file: str_66):  BOOLEAN;
VAR
   fil : FILE;

BEGIN
  Assign(fil, this_file);
  {$I-}
  RESET(fil);
  {$I+}
  exist := (IOResult = 0);
  CLOSE(fil);
END;
```

```
{******************************************************************************
*  GETX;  TAKES A PICTURE OF THE USER SCREEN                                  *
******************************************************************************}
FUNCTION getx(cl,LN, ncol, nln: INTEGER): rayptr;
VAR
  i,                     {Loop control variable}
  j,                     {Loop control variable}
  numbytes,              {number of bytes}
  start      : INTEGER; {start position}
  ray        : rayptr;  {position on screen pointer}
  screen_mem : ARRAY [1..4000] OF Byte Absolute $B800:0000;

BEGIN
  NEW(ray);
  start := ((LN-1) * 160) + ((cl * 2) - 1);
  numbytes := (ncol * 2);
  j := 1;
  FOR i := 1 TO nln DO
    BEGIN
      Move(screen_mem[start],ray^.pixray[j],numbytes);
      start := start + 160;
      j := j + numbytes;
    END;
  ray^.nc := ncol;
  ray^.nl := nln;
  getx := ray;
END;

{******************************************************************************
*  PUTX;  RETURNS PICTURE OF THE USER SCREEN BUT SAVES THE IMAGE              *
******************************************************************************}
PROCEDURE putx(cl,LN: INTEGER; ray: rayptr);
VAR
  i,                     {Loop control variable}
  j,                     {Loop control variable}
  numbytes,              {number of bytes}
  start       :INTEGER; {start position}
  screen_mem2 : ARRAY[1..4000] OF Byte Absolute $B800:0000;

BEGIN
  start := ((LN-1) * 160) + ((cl * 2) - 1);
  numbytes := (ray^.nc * 2);
  j := 1;
  FOR i := 1 TO ray^.nl DO
    BEGIN
      Move(ray^.pixray[j],screen_mem2[start],numbytes);
      start := start + 160;
      j := j + numbytes;
    END;
  if xfree then
     dispose(ray);         { release the image after signaling user }
  xfree := false;
END;
```

```
{******************************************************************************
*  CHK_MSG;  FINDS THE NUMBER OF MESSAGES ON THE QUEUE   (assume no deletes) *
******************************************************************************}

PROCEDURE chk_msg;

BEGIN
  done:=FALSE;
  msgint:=1;                                 {start at message 1}
  REPEAT
    STR(msgint,msgnum);
    newfile := usr_path+usrno+new_ext+msgnum;
    IF exist(newfile) THEN msgint:=msgint+1  {count up if it exists}
    ELSE
      BEGIN
        done:=TRUE;                          {set flag to exit}
        STR(msgint-1,msgnum);                {now have # msgs on queue}
      END;                                   {in msgnum and next msg #}
  UNTIL done;                                {in msgint}
  done:=FALSE;
END;

{*****************************************************************************
*   CHK_NUM;  FINDS THE ACTUAL NUMBER OF MESSAGES IN THE QUEUE               *
*****************************************************************************}

PROCEDURE chk_num;
BEGIN
  done := false;
  num := 1;                                  {start at message 1}
  REPEAT
    STR(num,msgnum);                         {convert num to string of 2}
    numfile := usr_path+usrno+new_ext+msgnum;
    IF exist(numfile) THEN num := num+1      {is there a message[num]?}
    ELSE  done := true;                      {if so, increment counter}
  UNTIL done;
  done := false;
END;

{----------------------------------------------------------------------------}
{                      NOW BEGINS THE REAL PROGRAM                           }
{----------------------------------------------------------------------------}

{*************************************************************************
*  MAIL_MAN;  NOTIFIES THE RECEIVER THAT HE HAS MESSAGES WAITING         *
*************************************************************************}

PROCEDURE mail_man;
VAR
  tics,
  hiword,
  loword    : REAL;
  hours,
  mins,
  secs      : STRING[2];
  time      : STRING[10];
  countdown : INTEGER;

    (*****************************************************************
    *  BEBEEP;  RINGS A BELL TO NOTIFY RECEIVER THAT HE HAS MSGS     *
    *           LOCAL PROCEDURE                                      *
    ******************************************************************)
    PROCEDURE bebeep;
    VAR n : Byte;

    BEGIN
      NoSound;
      FOR n := 1 TO 3 DO
        BEGIN
          Sound(800); Delay(50);
          Sound(400); Delay(50);
        END;
      NoSound;
    END;

BEGIN
  WHILE KeyPressed DO READ(Kbd,keychr);     {clear any waiting keys}
  IF (status AND timer_on) = timer_on THEN  {If our timer is ticking}
    BEGIN
      IF (status AND from_timer) = from_timer THEN  {and the timer finished}
        BEGIN                                       {then clear timer request}

          status := status AND NOT (timer_on + from_timer);
          Chk_num;                          {how many actual messages are there?}
          if (num < msgint) then            {have any been read since last time?}
             msgint := num;                 {if so, change the msg index}
          STR(msgint,msgnum);               {convert message number to string}
          STR(send_id,usrno);               {convert user number to string}
          newfile:=usr_path+usrno+new_ext+msgnum;   {get name of msg wanted}
          IF exist(newfile) THEN            {Is there new mail?}
            BEGIN                           {If new mail, tell the user.}
              chk_msg;                      {call to check number of messages}
              bebeep;                       {ring a bell}
              mkwin(5,7,75,13,black,green,3);       { Make a window}
              ClrScr;
              GotoXY(21,2);
              WRITELN('***** NEW MAIL *****');
              GotoXY(18,3);
              WRITELN(msgnum,' NEW MESSAGE(S) WAITING');
              GotoXY(13,5);
              WRITELN('ENTER SHIFT-F7 TO READ/SEND MESSAGES');
              gotoxy(48,4);
              Delay(2000);                  {let the user read the message}
              rmwin;                        {remove the window}
            END;
          STR(send_id,usrno);
          newnote := usr_path+usrno+note_ext;
          if exist(newnote) then
      BEGIN
       assign(note,newnote);
       reset(note);
       bebeep;
       mkwin(5,8,74,12,blue,cyan,1);
       clrscr;
       tpos := 1;
       while not eof(note) do
         begin
           readln(note,a_line);
           gotoxy(3,tpos);
           writeln(a_line);
           tpos := tpos + 1;
         end;
       close(note);
       gotoxy(3,tpos);
       textcolor(white+blink);
       write('            Press C to continue ');
       gotoxy(33,tpos);
       inp_char := 'x';
       while (inp_char <> 'C') and (inp_char <> 'c') do
         read(kbd,inp_char);
       rmwin;
       erase(note);
      END;
     set_timer(timer_time);      {restart the timer}
    END;
  END;
END;


{---------------------------------------------------------------------------}
{                THE ABOVE ARE THE USER INCLUDE ROUTINES                      }
{---------------------------------------------------------------------------}

{---------------------------------------------------------------------------}
{                     P R O C E S S   I N T E R R U P T                        }
{---------------------------------------------------------------------------}
{ PURPOSE:
   The following procedures  displace  standard interrupts.
   Do not put Variables or Constants in this Procedure. It will
   cause registers to be clobbered during the Interrupt routine
   when Turbo attempts to allocate  storage for local variables
   or parameters.}

PROCEDURE stay_int16;      {keyboard Interrupt 16 Service Routine}

   {If anything but "Our_HotKey" is pressed,  the key is  passed
   to the standard  keyboard service routine.  B_U_T,  when Our
   HotKey is recognized, a hotkey bit is set.}

BEGIN
{$I b:STAYI16.410}
END; {STAY_INT16}
```

```
PROCEDURE stay_int13;       {BIOS Disk interrupt  Routine}
BEGIN                       {Sets a flag while disk is active}
{$I b:STAYI13.410}
END; {STAY_INT13}

PROCEDURE stay_int21;       {DOS interrupt 21 Service Routine}
BEGIN                       {Sets a flag while INT 21 is active}
{$I b:STAYI21.410}
END; {STAY_INT21}

PROCEDURE stay_int8;        {Timer Interrupt 8 Service Routine}
                            {Activates Stayres during pgm execution}
BEGIN                       {when safe to do so.}
{ add the following include for the Clock Demonstration by Neil Rubenking }
{$I b:CLKI8.410}
{$I b:STAYI8.420}
END;{Stay_Int8}


PROCEDURE stay_int28;       {Idle Interrupt 28 Service Routine}
BEGIN                       {Invokes Stayres from the DOS prompt}
{$I b:STAYI28.410}
END;{Stay_Int28}            {continue}

PROCEDURE staysave;         {Prolog to Resident Turbo Code}
BEGIN
{$I b:STAYSAVE.420}
  getdta(saveddta[1],saveddta[2]);        {Save callers DTA address}
  getpsp(savedpsp);                       {Save callers PSP Segment}
  setpsp(ourpsp);                         {Set our PSP Segment}
  setdta(ourdta[1],ourdta[2]);            {Set our DTA address}
  newctlc[2] := CSeg;
  newctlc[1] := Ofs(iret);
  getctlc(savedctlc); setctlc(newctlc);   {Get/Save the users Ctrl-C vector
  int24on;                                {Trap Dos Critical Errors}

      {-----------------------------------------------------
      {             INVOKE USER PROCEDURE HERE
      {----------------------------------------------------

       BEGIN
         xp := wherex;          {get the main x cursor [illegible]
         yp := wherey;          {get the main y cursor [illegible]
         pic := getx(1,1,80,25);{take a picture [illegible]
         keychr := #0;          {clear any [illegible]
         mail_man;
         xfree := true;
         putx(1,1,pic);
         gotoxy(xp,yp);         [illegible]
       END;

      {-------
      {             EN[illegible]
```

AD-A185 758   OPTIMIZATION ON ENHANCEMENT OF THE PLEXSYS NETWORK   2/3
MESSAGING SYSTEM (EMAIL)(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH G D MCALUM 1987
UNCLASSIFIED   AFIT/CI/NR-87-76T   F/G 25/5   NL


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

```
  setpsp(savedpsp);                     { Restore Callers PSP Segment}
  setdta(saveddta[1],saveddta[2]);      { Restore the users DTA}
  setctlc(savedctlc);                   { Restore the users Ctrl-C Vector}
  int24off;                             { Remove Our Critical Error routine}


{----------------------------------------------------------------------------}
{                 BEGINNING OF THE STAYRSTR ROUTINE                          }
{----------------------------------------------------------------------------}
{$I b:STAYRSTR.420}

          {----------------------------------------------------}
          {        END OF THE STAYRSTR ROUTINE                 }
          {----------------------------------------------------}

END;
{----------------------------------------------------------------------------}
{                              M  A  I  N                                    }
{----------------------------------------------------------------------------}
          { The main program installs the new interrupt routine }
          { and makes it permanently resident as the keyboard   }
          { interrupt.  The old keyboard interrupt Vector is    }
          { stored in Variables , so they can be used in Far    }
          { Calls.                                              }
          {                                                     }
          { The following dos calls are used:                  }
          { Function 25 - Install interrupt address             }
          {               input al = int number,                }
          {               ds:dx = address to install            }
          { Function 35 - get interrupt address                 }
          {               input al = int number                 }
          {               output es:bx = address in interrupt   }
          { Function 31 - terminate and stay resident           }
          {               input dx = size of resident program   }
          {               obtained from the memory              }
          {               allocation block at [Cs:0 - $10 + 3]  }
          { Function 49 - Free Allocated Memory                 }
          {               input Es = Block Segment to free      }
          {-----------------------------------------------------}

BEGIN
  ourdseg:= DSeg;               {Save the Data Segment Address for Interrupts}
  oursseg:= SSeg;               {Save our Stack Segment for Interrupts}
  getpsp(ourpsp);               {Local PSP Segment}
  getdta(ourdta[1],ourdta[2]);  {Record our DTA address}
  userprogram:=Ofs(staysave);   {Set target of call instruction}
  regs.ax := $3000 ;            {Obtain the DOS Version number}
  Intr(dosi21,regs);
  dosversion := halfregs.al;    { 0=1+, 2=2.0+, 3=3.0+ }
                                {Obtain the DOS Indos status location}
  regs.ax := $3400;
  Intr(dosi21,regs);
  dosstat1.ip  := regs.bx;
  dosstat1.cs  := regs.es;
  dosstat2.cs  := regs.es;
  dossseg      := regs.es;
  bytecount := 0;              {Search for CMP [critical flag],00 instruct.}
    WHILE (bytecount < $2000)  (then Mov SP,stackaddr instruction }
      AND (MemW[dosstat2.cs:bytecount] <> $3e80)
      DO bytecount := SUCC(bytecount);
      IF bytecount = $2000 THEN      { Couldn't find critical flag addr }
        BEGIN
          WRITELN('StayRes incompatiblity with Operating System');
          WRITELN('StayRes will not install correctly..Halting');
          HALT;
        END;

      { Search for the DOS Critical Status Byte address.      }
      { Bytecount contains offset from DosStat1.CS of the     }
      {        CMP [critical flag],00                         }
      {        JNZ ....                                       }
      {        Mov SP,indos stack address                     }

      IF Mem[dosstat2.cs:bytecount+7] = $bc THEN   {MOV SP,xxxx}
        BEGIN
          dosstat2.ip := MemW[dosstat2.cs:bytecount+2];
          dossptr     := MemW[dosstat2.cs:bytecount+8]; {INDOS Stack address}
        END
      ELSE
        BEGIN
           WRITELN('Cannot Find Dos Critical byte...Halting');
           HALT;
        END;
      InLine($fa);                       {Disable interrupts}


      { Setup Our Interrupt Service Routines }

      setup_interrupt(biosi8, bios_int8, Ofs(stay_int8));    {timer}
      setup_interrupt(biosi13, bios_int13, Ofs(stay_int13)); {disk}
      setup_interrupt(dosi21, dos_int21, Ofs(stay_int21));   {DOSfunction}
      setup_interrupt(dosi28, dos_int28, Ofs(stay_int28));   {DOS idle}
      InLine($fb);                        {Re-enable interrupts}

{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
{------------------------------------------------------------------------------}
{                     INITIALIZE YOUR PROGRAM HERE                             }
{------------------------------------------------------------------------------}
{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}

      { Initialize Program Here since we will not get control again
        until "Our_HotKey" is entered from the Keyboard.              }
      set_timer (timer_time);     {start the first timer}
      Assign(user,usr_id_str);
      RESET(user);
      READLN(user,send_id);       {get user's number}
      CLOSE(user);
      done := FALSE;
      xfree := FALSE;
      msgint:=1;                   {look for message number 1}

     {-------------------------------------------------------------}
     {              END OF INITALIZE PROGRAM CODE                  }
     {-------------------------------------------------------------}

   { Now stay resident.   The following Call utilizes the DOS Stay Resident
     We get the amount of memory by fetching the memory allocation paragraphs
     from the Memory Control Block.  This was set by Turbo initialization during
     Int 21/function 4A (shrink block), calculated from the mInimum and mAximum
     options menu. The MCB sits one paragraph above the PSP. }

                                         { Pass return code of zero    }
     regs.ax := $3100 ;                  { Terminate and Stay Resident }
     regs.dx := MemW [CSeg-1:0003]+1 ;   { Prog_Size from Allocation Blk}
     Intr (dosi21,regs);

     { END OF RESIDENCY CODE }
END.
```

APPENDIX C

PROGRAM LISTING

of

LOGON.PAS

```
{*****************************************************************************}
{* This program is called by GOMAIL.BAT and is used to allow a user to enter *}
{* their own username prior to loading the EMAIL system into resident memory.*}
{* It handles all error checking before making any entries to IDFILE.DAT, the*}
{* file which contains all usernames and node numbers for the network.       *}
{*****************************************************************************}

{$C-}

Program Logon;

 CONST
    nodes = 16;
    flag_fil   : string[66] = 'E:FLAG.FIL';      { used by Lock and Unlock}
    temp_fil   : string[66] = 'E:TEMP.FIL';      { used by Lock and Unlock}
    user_id    : string[66] = 'D:\MAIL\USER.ID'; { contains user's id number}
    idfile_dat : string[66] = 'E:IDFILE.DAT';    { file for mapping id's/names}

 TYPE
    entry_name = string[20];               { random access file, IDFILE.DAT }
    entry = record
              Uname : entry_name;
              id    : integer;
            end;
    st_20  = string[20];
    str_66 = string[66];

 VAR
     name             : string[20];
     idlist           : file of entry;
     an_entry         : entry;
     send_id,t,x,ctr  : integer;
     temp,ordval      : integer;
     go_val           : char;
     unlocked,
     empty,done,told  : boolean;
     right, dupe      : boolean;
     tempfile,                                 { assigned to TEMP.FIL }
     flagfile,                                 { assigned to FLAG.FIL }
     user             : text;                  { assigned to USER.ID  }


{*************************************$************************************}
{      EXIST;  RETURNS TRUE IF this_file EXISTS IN THE DIRECTORY           }
{**************************************************************************}

FUNCTION exist(this_file: str_66):  BOOLEAN;
VAR
   fil : FILE;

BEGIN
  Assign(fil, this_file);
  {$I-}
  RESET(fil);
  {$I+}
  exist := (IOResult = 0);
  CLOSE(fil);
END;

{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
{ This procedure is used to rename a file called FLAG.FIL to a file called }
{ TEMP.FIl.  This occurs when IDFILE.DAT is being accessed.  In effect, it }
{ "locks out" IDFILE.DAT from the other processes.  If FLAG.FIL does not   }
{ exist, signifying that IDFILE.DAT is already in use, this procedure will }
{ go into a loop until IDFILE.DAT is available to be locked out.           }
{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}

Procedure lock;
 var go : boolean;
 begin
  assign(tempfile,temp_fil);
  assign(flagfile,flag_fil);
  unlocked := true;
  REPEAT
   if  exist(flag_fil) then
     begin
       {$I-}
       rename(flagfile,temp_fil);
       {$I+}
       go := (IOresult = 0);
       if go then
         begin
           close(tempfile);
           unlocked := false;
         end;
     end
   else
     begin
       delay(1000);
     end;
  UNTIL not unlocked;
 end;


{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
{ This procedure is used to "unlock" IDFILE.DAT.  It does this by renaming }
{ the file TEMP.FIL to FLAG.FIL.  It also sets a flag called unlocked to   }
{ true.  This flag is used by Procedure Lock.                              }
{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}

Procedure unlock;
 begin
  unlocked := true;
  rename(tempfile,flag_fil);
  close(flagfile);
 end;
```

```
{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
{ THIS PROCEDURE IS USED BY THE MAIN PROGRAM TO CHECK AN ENTERED USERNAME FOR }
{ SPACES ANYWHERE WITHIN THE LENGTH OF THE NAME THAT IT IS ENTERED. UP TO 20. }
{ IF AT LEAST ONE SPACE IS FOUND, IT WILL DISPLAY AN ERROR MESSAGE AN REQUEST }
{ A REENTRY.  NO OTHER PROCEDURES ARE CALLED UNTIL A CORRECT NAME IS ENTERED. }
{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}

Procedure space_chk(name2 : st_20);
  var
      i       : integer;
      name3   : string[20];
begin
 t := 0;
 right := false;
 name3 := '                    ';
 t :=length(name2);
 for i := 1 to t do
    name3[i] := name2[i];
 i := 1;
 while (name3[i] <> ' ') and (i <> t) and (t <> 0) do
     i := i+1;
 if (i <> t) and (t <> 0) then
   i := i+1;
 if (name3[i] = ' ') and (name3[1] <> ' ') and (t <> 0) then
   right := false
 else
   begin
    if (name3[1] <> ' ') and (name3[i-1] <> ' ') or (t = 0) then
      right := true
     else
        ;
   end;
 if not right then
   begin
     writeln('NO SPACES ALLOWED, PLEASE REENTER');
   end;
end;  { end Procedure space_chk }


{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
{ THIS PROCEDURE IS USED TO CONVERT ALL CHARACTERS OF THE ENTERED NAME TO UP- }
{ PER CASE.  A WORKING STRING IS USED TO COPY THE CONVERTED CHARACTERS TO. IF }
{ A CHARACTER IS ALREADY ENTERED IN UPPERCASE, IT WILL NOT BE AFFECTED BY THE }
{ THE CONVERSION. ONLY CHARACTERS A-Z ARE CONSIDERED VALID ENTRIES.           }
{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}

Procedure ch_case(var name2 : st_20);
var
  name3    : string[20];
  stop           : boolean;
  i, val         : integer;

begin
  name3 := '                    ';
  stop := false;
  i := 1;
  repeat
   val := ord(name2[i]);
   if ((val<=122) and (val>=97)) or ((val)=65) and (val<=90)) then
      begin
        name3[i] := upcase(name2[i]);
        i := i + 1;
      end
   else
      stop := true;
  until stop;
  name2 := name3;
  end;  { end Procedure ch_case }


{******************************************************************************}
{ THIS IS THE ACTUAL PART OF THE PROGRAM WHICH PROMPTS THE USER FOR EACH USER-}
{ NAME.  IT ALSO DOES THE NECESSARY ERROR CHECKING FOR EACH INPUT.  THE USER  }
{ IS NOT REQUIRED TO ENTER A NAME FOR EVERY NODE.  A CARRIAGE RETURN SIGNIFIES}
{ NO ONE IS ASSIGNED TO THAT POSITION ON THE NETWORK.  THE USER HAS THE OPPOR-}
{ TUNITY TO ADD A NAME AT A LATER POINT HOWEVER.                               }
{******************************************************************************}

Procedure Main;
 begin
  empty := true;
  REPEAT
    REPEAT
       name := '                     ';
       writeln;
       write('Enter your username  ---> ');
       readln(name);
       space_chk(name);
    if right then
     begin
       temp := length(name);
       x := 1;
       REPEAT
         ordval := ord(name[x]);
         if ((ordval<=122)and(ordval>=97))or((ordval)=65)and(ordval<=90)) then
           x := x + 1
         else
           begin
             if (ordval <> 32) then
               begin
                writeln('INVALID CHARACTER, PLEASE REENTER');
                right := false;
               end;
           end;
       UNTIL (x = temp+1) or not right;
    end;
    UNTIL right;
    ch_case(name);
    if (t = 0) then
      writeln('You must enter a username, TRY AGAIN!')
    else
      empty := false;
  UNTIL not empty;
 end; { end procedure Main }


{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
{ This procedure contains the primary loop for entering a username.  It calls }
{ procedure main which accomplishes all error checking on the name which is   }
{ input by the user.  This procedure will also give the user a chance to mod- }
{ their username, before entering into the duplicate-username check.          }
{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}

Procedure inp_name;
 begin
   done := false;
   REPEAT
      right := false;
      Main;
      go_val := 's';
      while (go_val <> 'N') and (go_val <> 'n') do
        begin
          writeln;
          writeln('You have entered the following username ----> ',name);
          writeln;
          write('Do you want to change it? (Y/N) ---> ');
          readln(go_val);
          if (go_val = 'Y') or (go_val = 'y') then
            begin
              right := false;
              main;
            end;
        end;
       if right then done := true;
   UNTIL DONE;
   done := false;
 end;

{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$  End Procedures $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}


 begin
   writeln;
   writeln;
   textcolor(white);
   assign(user,user_id);
   reset(user);
   readln(user,send_id);
   close(user);
   inp_name;                       { call Procedure Inp_name }
   assign(idlist,idfile_dat);
   lock;
```

```
    REPEAT
       reset(idlist);
       dupe :=  false;
       ctr := 1;
       while (ctr in [1..nodes]) and (not dupe) do
         begin
           seek(idlist,ctr-1);
           read(idlist,an_entry);
           with an_entry do
             begin
              if (send_id <> id) then
                begin
                 if (name = uname) then    { is name a duplicate? }
                   dupe := true;           { if yes, set flag }
                end;
             end;
           ctr := ctr + 1;
         end;
       if not dupe then
         begin
           done := true;
           seek(idlist,send_id-1);
           read(idlist,an_entry);
           with an_entry do                { if not a duplicate then }
             begin                         { assign username to proper }
              uname := name;               { entry position in IDFILE.DAT }
              id    := send_id;
             end;
           seek(idlist,send_id-1);
           write(idlist,an_entry);
         end
       else
         begin
           writeln;
           writeln('Duplicate username found, please enter another name.');
           unlock;
           inp_name;
           lock;
         end;
       close(idlist);
    UNTIL DONE;
 unlock;
 writeln;
 writeln;
 writeln('**************************************');
 writeln('*  Name entered into user directory  *');
 writeln('**************************************');
 writeln;
end.
{$C+}
```

APPENDIX D

PROGRAM LISTING

of

SEND.PAS

```
{*****************************************************************************}
{* This program is used to send an "on-screen" message to any user on the  *}
{* the network.  It can only be invoked from the DOS prompt.  After the    *}
{* user types "SEND", they will be prompted for the receiver's username.   *}
{* This name is checked against IDFILE.DAT to see if it is valid.  Once a  *}
{* name is correct and valid, the user will be prompted for the message    *}
{* which cannot exceed 65 characters in length. After the message has been *}
{* created the program places the message into a temporary file called     *}
{* TEMP.MES which is located in the user's subdirectory MAIL. The program  *}
{* will then attempt to place is in the receiver's MAIL subdirectory if no *}
{* other message is being read.  Note:  there is no queueing function. The *}
{* program will simply loop until there is an available slot to place the  *}
{* message.                                                                 *}
{*****************************************************************************}

{$C-}

Program Send(input,output);

CONST
   nodes      = 16;                     {number of stations on the network}
TYPE
   entry_name = string[20];
   entry      = record                  {random access file, IDFILE.DAT}
                  name   : entry_name;
                  id     : integer;
                end;

   str_66     = string[66];
   str_20     = string[20];
   str_4      = string[4];
   user_table = array[1..nodes] of str_66;
   filename   = string[66];
   Tx_String  = String[72];

CONST
   prompt1     =  'Enter the receiver's username  ---> ';
   prompt2     =  'Enter your note below; limit 65 characters.';
   from        =  'MESSAGE FROM:  ';
   user_id_str:  str_66 = 'D:\MAIL\user.id';    {contains user id number}
   userx      :  str_4  = 'user';               {user extension}
   idfile_dat :  str_66 = 'D:\MAIL\idfile.dat'; {contains all usernames/id#'s}
   temp_file  :  str_66 = 'D:\MAIL\temp.mes';   {holds send message temporarily}
   flag_fil   :  str_66 = 'D:\MAIL\flag.fil';   {used by Lock and Unlock}
   temp_fil   :  str_66 = 'D:\MAIL\temp.fil';   {used by Lock and Unlock}
   note_ext   :  str_66 = '.mes';               {message extension}
   LArr       = 75;
   RArr       = 77;
   Del        = 83;
   Escape     = 27;
   Bksp       = 8;
   Enter      = 13;
   Ins        = 82;
   user_filetable: user_table =         { user path table used in PUT.DIR }
    ('E:\USER1\MAIL\',                  { procedure to enable file copying }
     'E:\USER2\MAIL\',                  { from one node directory to another }
     'E:\USER3\MAIL\',
     'E:\USER4\MAIL\',
     'E:\USER5\MAIL\',
     'E:\USER6\MAIL\',
     'E:\USER7\MAIL\',
     'E:\USER8\MAIL\',
     'E:\USER9\MAIL\',
     'E:\USER10\MAIL\',
     'E:\USER11\MAIL\',
     'E:\USER12\MAIL\',
     'E:\USER13\MAIL\',
     'E:\USER14\MAIL\',
     'E:\USER15\MAIL\',
     'E:\USER16\MAIL\');

{ Remove the comment brackets and append the following user table entries }
{ to the above user table so that software is setup to handle 32 nodes on }
{ the network.  Currently, the system is set up for 16 user nodes.       }

{     'E:\USER17\MAIL\',
      'E:\USER18\MAIL\',
      'E:\USER19\MAIL\',
      'E:\USER20\MAIL\',
      'E:\USER21\MAIL\',
      'E:\USER22\MAIL\',
      'E:\USER23\MAIL\',
      'E:\USER24\MAIL\',
      'E:\USER25\MAIL\',
      'E:\USER26\MAIL\',
      'E:\USER27\MAIL\',
      'E:\USER28\MAIL\',
      'E:\USER29\MAIL\',
      'E:\USER30\MAIL\',
      'E:\USER31\MAIL\',
      'E:\USER32\MAIL\');  }


VAR
   user          :  text;        { contains user's id number, USER.ID }
   temp          :  text;        { temporary storage for the message }
   notefile      :  text;        { destination file for message }
   flagfile      :  text;        { FLAG.FIL, used to lock }
   tempfile      :  text;        { TEMP.FIL, used to lock }
   the_note      :  string[80];  { contains the actual message }
   a_line        :  string[80];  { one line in a file, 80 chars. }
   recvfile      :  filename;    { concatenated destination file name }
   send_id,                      { sender's id number }
   ctr,                          { counter for accessing IDFILE.DAT }
   recv_id,                      { receiver's id number }
   t,v,i,                        { temporary variables }
   numpram        :  integer;     { number of parameters used by sender }
   userno         :  string[2];
   an_entry       :  entry;       { one record from IDFILE.DAT }
   idlist         :  file of entry;  { random access file }
   from_name      :  str_20;      { sender's username }
   to_name        :  str_20;      { receiver's username }
   param          :  str_20;      { parameter, if used }
   from_string    :  string[35];
   told,pram,qt,
   go, ok, found  :  boolean;
   right          :  boolean;
   ready,unlocked :  boolean;




{########################  Begin Procedures  #########################}

{---------------------------------------------------
=  GETSTRING;  Gets a string input from keyboard.     =
=     The parameters col and lne specify the starting =
=   coordinates on the full (1,1,80,25) screen. Before=
=   calling GetString, do a gotoxy(x,y) command to    =
=   place the cursor within whatever window is active.=
---------------------------------------------------}
Function GetString(Len: integer; In_str: Tx_String): Tx_String;

Var
  ch       : char;
  lnest,
  colst,
  col,lne,
  i,
  colindex : integer;
  blankln  : Tx_String;
  insertmode : boolean;

Begin
  blankln := '';
  for i := 1 to Len do
    blankln := blankln + ' ';
  insertmode := true;
  colst := wherex;
  col := colst;
  colindex := 1;
  lnest := wherey;
  lne := lnest;
  gotoxy(colst,lnest);
  Write(In_str);
  ch := ' ';
  while (Ord(ch) <> Enter) do
    begin
      gotoxy(col,lne);
```

```
read(kbd,ch);
  if (Ord(ch) > 31) and (Ord(ch) < 126) then
    begin
      if not insertmode then
        begin
          if colindex <= Len then
            begin
              Delete(In_str,colindex,1);
              Insert(ch,in_str,colindex);
              colindex := succ(colindex);
              col := succ(col);
            end;
        end
      else
        if Length(in_str) < Len then
          begin
            Insert(ch,in_str,colindex);
            colindex := colindex + 1;
            col := col + 1;
          end;
      gotoxy(colst,lnest);
      Write(in_str);
    end
  else
    if (Ord(ch) = Escape) and (keypressed) then
      begin
        read(kbd,ch);
        case Ord(ch) of
          LArr : if colindex > 1 then
                   begin
                     colindex := colindex - 1;
                     col := col - 1;
                   end;
          RArr : if (colindex <= Length(in_str)) then
                   begin
                     colindex := colindex + 1;
                     col := col + 1;
                   end;
          Del : begin
                  Delete(In_Str,colindex,1);
                  gotoxy(colst,lnest);
                  write(blankln);
                  gotoxy(colst,lnest);
                  write(In_str);
                end;

          Ins : insertmode := not insertmode;
        end;
      end
    else
        if Ord(ch) = Bksp then
          begin
            if colindex > 1 then
              begin
```

```
                      colindex := colindex - 1;
                      col := col - 1;
                      Delete(In_Str,colindex,1);
                      gotoxy(colst,lnest);
                      write(blankln);
                      gotoxy(colst,lnest);
                      write(In_str);
                    end;
                end
              else
                if Ord(ch) = escape then
                  if Length(In_Str) > 0 then
                    begin
                      In_str := '';
                      gotoxy(colst,lnest);
                      write(blankln);
                      col := colst;
                      colindex := 1;
                    end
                  else
                    begin
                      ch := Chr(Enter);
                      In_Str := 'ESCAPE';
                    end;
    end;
  GetString := In_Str;
End;


{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
{    EXIST;  returns true if this_file exists in the directory            }
{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}

FUNCTION exist(this_file: str_66):  BOOLEAN;
VAR
   fil : FILE;

BEGIN
  Assign(fil, this_file);
  {$I-}
  RESET(fil);
  {$I+}
  exist := (IOResult = 0);
  CLOSE(fil);
END;

{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
{                             Procedure LOCK                               }
{ This procedure is used to rename a file called FLAG.FIL to a file called }
{ TEMP.FIl.  This occurs when IDFILE.DAT is being accessed.  In effect, it }
{ "locks out" IDFILE.DAT from the other processes.  If FLAG.FIL does not   }
{ exist, signifying that IDFILE.DAT is already in use, this procedure will }
{ go into a loop until IDFILE.DAT is available to be locked out.           }
{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
```

```
Procedure lock;
 var go : boolean;
 begin
  assign(tempfile,temp_fil);
  assign(flagfile,flag_fil);
  unlocked := true;
  REPEAT
   if  exist(flag_fil) then
     begin
       {$I-}
       rename(flagfile,temp_fil);
       {$I+}
       go := (IOresult = 0);
       if go then
         begin
           close(tempfile);
           unlocked := false;
         end;
     end
   else
     begin
       delay(500);
     end;
  UNTIL not unlocked;
 end;


{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
{                           Procedure UNLOCK                               }
{ This procedure is used to "unlock" IDFILE.DAT.  It does this by renaming }
{ the file TEMP.FIL to FLAG.FIL.  It also sets a flag called unlocked to   }
{ true.  This flag is used by Procedure Lock.                              }
{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}

Procedure unlock;
 begin
  unlocked := true;
  rename(tempfile,flag_fil);
  close(flagfile);
 end;



{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
{  ERROR_CHK;  checks for any invalid chars. or spaces in the user name  }
{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}

PROCEDURE error_chk(name2 : str_20);

VAR
  name     : STRING[22];           {temp storage for receiver's name}
```

```
BEGIN
  right := FALSE;
  name := '                    ';
  t :=LENGTH(name2);
  FOR i := 1 TO t DO
     name[i] := name2[i];
  i := 1;
  v := 0;
  FOR i := 1 TO t DO
    BEGIN
      IF (name[i] = ' ') THEN
        v := v+1;
    END;
  IF (v = 0) THEN
      right := TRUE
  ELSE
      BEGIN
        writeln;
        writeln;
        WRITELN('NO SPACES ALLOWED, PLEASE REENTER');
        WRITELN;
      END;
  IF right THEN
    BEGIN
      i := 1;
      REPEAT
        v := ORD(name[i]);
        IF((v<=122)AND(v>=97))OR((v)>=65)AND(v<=90)) THEN
          i := i + 1
        ELSE
          BEGIN
            writeln;
            writeln;
            WRITELN('INVALID CHARACTER, PLEASE REENTER');
            right := FALSE;
          END;
      UNTIL (i = t+1) OR NOT right;
    END;
END;

{##########################################################################}
{  CH_CASE;  converts receiver's name to uppercase before comparing       }
{            it to user names in IDFILE.DAT.                               }
{##########################################################################}

PROCEDURE ch_case(VAR name2 : str_20);
VAR
  name      : STRING[20];
  tp        : INTEGER;

BEGIN
  name := '                    ';
  i := 1;
  tp := LENGTH(name2);
  REPEAT
    name[i] := UpCase(name2[i]);
    i := i + 1;
  UNTIL i = tp + 1;
  name2 := name;
END;

{*******************************************************************}
{  CHK_USER;  checks to see if name entered as receiver is a valid  }
{           name.                                                   }
{*******************************************************************}

PROCEDURE chk_user(name2 : str_20);

VAR
   temp_name : STRING[20];

BEGIN
   found := FALSE;
   Assign(idlist,idfile_dat);
   RESET(idlist);
   ctr := 1;
   while (ctr in [1..nodes]) and (not found) do
      begin
        seek(idlist,ctr-1);
        read(idlist,an_entry);
        with an_entry do
         begin
           if (name2 = name) then
             begin
              found := true;
              recv_id := id;
             end;
         end;
        ctr := ctr + 1;
      end;
   CLOSE(idlist);
   IF NOT found then
     begin
       pram := false;
       writeln;
       writeln;
       writeln('USER NAME NOT FOUND, TRY AGAIN');
     end;
   WRITELN;
 END;

{***************************  End Procedures  *****************************}

begin
  textcolor(white);
  assign(user,user_id_str);
  reset(user);                  { get user's id number from USER.ID }
  readln(user,send_id);
 close(user);
 assign(idlist,idfile_dat);
 lock;
 reset(idlist);
 ctr := send_id - 1;
 seek(idlist,ctr);
 read(idlist,an_entry);         { get user's username from IDFILE.DAT }
 with an_entry do
    begin
      from_name := name;
    end;
 close(idlist);
 unlock;
 qt := false;
 pram := false;
 found := false;
 right := false;
 numpram := paramcount;
 if numpram > 0 then             { were parameters passed? }
   begin
     pram    := true;
     to_name := paramstr(1);
     right   := true;
   end;
 REPEAT
   if not pram then
     begin
       writeln;
       write(prompt1);             { prompt the user for receiver's username }
       to_name := getstring(20,'');
       if (to_name = 'ESCAPE') then
          qt := true
       else
          error_chk(to_name);         { check input name for correctness }
     end;
   if (right) and (not qt) then
     begin
       ch_case(to_name);       { change name to uppercase }
       lock;
       chk_user(to_name);      { check to see if valid username, and get id number }
       unlock;
     end;
 UNTIL right and found or qt;
if not qt then
  begin
     ok := false;
     from_string := from + from_name;
     assign(temp,temp_file);
     rewrite(temp);
     writeln(temp,from_string);         { write header to temporary file }
     pram := false;
     the_note := ' ';
     if (numpram > 1) then
       begin
       pram := true;
       i := numpram;
       for i := 2 to numpram do
          begin
            param := paramstr(i);
            if (i = 2) then
               the_note := param
            else
               the_note := the_note+' '+param;
          end;
     end;
   REPEAT
     if not pram then
       begin
         writeln;
         writeln(prompt2);                  { prompt user for the message }
         the_note := getstring(65,'');
         if (the_note = 'ESCAPE') then
           begin
             qt := true;
             close(temp);
             erase(temp);
           end;
      end;
     if not qt then
       begin
         ok := true;
         writeln(temp,the_note);      { place the message in temporary file }
         writeln;
         writeln;
         writeln('******* MESSAGE BEING SENT ');
         close(temp);
         delay(100);
       end
    else
       ok := true;
   UNTIL ok;
if not qt then
  begin
   STR(recv_id,userno);
   recvfile := user_filetable[recv_id]+userx+userno+note_ext;
   ready := false;
   told := false;
   i := send_id * 100;
   REPEAT
    delay(i);
    if exist(recvfile) then       { is receiver already reading another message? }
      begin
        if not told then
          begin
            told := true;
            writeln;
            writeln('******* PAUSING, ANOTHER MESSAGE IS AHEAD OF YOURS');
          end;
```

```
           delay(500);
         end
       else
         begin
           if not exist(recvfile) then
             begin
               assign(notefile,recvfile);
               {$I-}
               rewrite(notefile);
               {$I+}
               go := (IOresult = 0);
               if go then
                begin
                  reset(temp);
                  ready := true;
                  while not eof(temp) do
                    begin
                      readln(temp,a_line);
                      writeln(notefile,a_line);       { copy message to receiver's dir.}
                    end;
                  close(notefile);
                  close(temp);
                  erase(temp);
                end;
             end;
         end;
        i := i + 100;
      UNTIL ready;
      writeln;
      writeln('$$$$$$$ MESSAGE RECEIVED ');
      writeln;
     end;
   end;
  writeln;
end.
{$C+}
```

APPENDIX E

PROGRAM LISTING

of

BLDFILE.PAS

```
{*************************************************************************}
{*  BLDFILE.PAS                                                         *}
{*                                                                       *}
{* THIS PROGRAM IS USED BY EINIT.BAT WHEN INITIALIZING THE EMAIL SYSTEM. *}
{* IT CREATES A RANDOM ACCESS FILE CALLED IDFILE.DAT WITH THE NUMBER OF  *}
{* ENTRIES CORRESPONDING TO THE VALUE OF THE CONSTANT NODES.  EACH ENTRY *}
{* IS INITIALIZED WITH BLANKS BUT WILL EVENTUALLY CONTAIN EACH PERSON'S  *}
{* USERNAME/NUMBER TO BE USED AS A MAPPING WHEN SENDING AND RECEIVING    *}
{* MAIL.  THE CREATED FILE IS LEFT IN THE PUBLIC DIRECTORY AND WILL BE   *}
{* ACCESSED BY EACH USER WHEN LOGGING ON AND UTILIZING THE EMAIL SYSTEM. *}
{*                                                                       *}
{*************************************************************************}

Program Bldfile;

 CONST
    nodes = 16;                    { number of stations on the network }

 TYPE
    entry_name = string[20];
    entry = record                 { random access file }
              name : entry_name;
              id   : integer;
            end;

 VAR
   idlist     : file of entry;
   an_entry   : entry;
   i          : integer;

 begin
   assign(idlist,'idfile.dat');
   rewrite(idlist);
   with an_entry do
   begin
     name := '                    ';
     for i := 1 to nodes do
      begin
        id := i;                   { initialize id#'s with 1 through nodes }
        write(idlist,an_entry);    { initialize names with blanks }
      end;
   end;
   close(idlist);
 end.
```

APPENDIX F

PROGRAM LISTING

of

LOCK.PAS

```
{****************************************************************************}
{ This program is used to rename a file called "FLAG.FIL" to a file called }
{ TEMP.FIl.  This occurs when IDFILE.DAT is being accessed.  In effect, it }
{ "locks out" IDFILE.DAT from the other processes.  If FLAG.FIL does not   }
{ exist, signifying that IDFILE.DAT is already in use, this program will   }
{ go into a loop until IDFILE.DAT is available to be locked out.           }
{****************************************************************************}

Program Lock;

 type
     str_66        = string[66];
 const
     idfile_dat    : str_66 = 'IDFILE.DAT';
     flag_fil      : str_66 = 'FLAG.FIL';
     temp_fil      : str_66 = 'TEMP.FIL';

 var go,
     unlocked      : boolean;
     tempfile,
     flagfile      : text;
     i             : integer;


{*****************************************************************************
*  EXIST;  RETURNS TRUE IF this_file EXISTS IN THE DIRECTORY                 *
******************************************************************************}
FUNCTION exist(this_file: str_66):  BOOLEAN;
VAR
   fil : FILE;

BEGIN
  Assign(fil, this_file);
  {$I-}
  RESET(fil);
  {$I+}
  exist := (IOResult = 0);
  CLOSE(fil);
END;

 begin
  assign(tempfile,temp_fil);
  assign(flagfile,flag_fil);
  unlocked := true;
  i := 1;
  REPEAT
   if exist(flag_fil) then
     begin
       {$I-}                         {compiler directive, turnoff run time check}
       rename(flagfile,temp_fil);
       {$I+}                         {compiler directive, turnon run time check }
       go := (IOresult = 0);
       if go then
```

```
        begin
          close(tempfile);
          unlocked := false;
        end;
    end
  else
    begin
      if (i = 1) then
        writeln('**** pausing, IDFILE.DAT in use ****');
      i := 2;
      delay(1000);
    end;
 UNTIL not unlocked;
end.
```

```
{**************************************************************************}
{ This program  is used to "unlock" IDFILE.DAT.   It does this by renaming }
{ the file TEMP.FIL to FLAG.FIL.  It also sets a flag called unlocked to   }
{ true.  This flag is used by program Lock.                                }
{**************************************************************************}

Program Unlock;

 const
     idfile_dat     : string[66] = 'IDFILE.DAT';
     flag_fil       : string[66] = 'FLAG.FIL';
     temp_fil       : string[66] = 'TEMP.FIL';
 var
     tempfile,
     flagfile       : text;

 begin
  assign(flagfile,flag_fil);
  assign(tempfile,temp_fil);
  rename(tempfile,flag_fil);
  close(flagfile);
 end.
```

APPENDIX G

PROGRAM LISTING

of

UNLOCK.PAS

APPENDIX H

PROGRAM LISTING

of

BLDMSG.INC

```
{*********************************************************************
*  BLDMSG.INC                                                        *
*                                                                    *
*  This include file allows a user to send a message to a receiver   *
*  or broadcast to everybody.  Various procedures check the validity *
*  of the input user name to ensure it is in the correct format and  *
*  that it is a valid name in idfile.dat.                            *
*                                                                    *
*********************************************************************}

PROCEDURE Bld_msg;

CONST
  messagea = 'Do you want to see the username list, (Y/N)? ';
  message1 = 'Receiver's username?  ("*" to broadcast/return to exit).';
  message2 = 'Please enter your message below.';
  wind1    = 1;

TYPE
  st_20  = STRING[20];


VAR
  i,t,v                 : INTEGER;     { temporary variables }
  right, found, ok      : BOOLEAN;
  listval               : CHAR;

{********************************************************************
*  ERROR_CHK;  CHECKS FOR ANY INVALID SPACES IN THE USER NAME       *
********************************************************************}
PROCEDURE error_chk(name2 : st_20);

VAR
  name     : STRING[22];            {temp storage for receiver's name}

BEGIN
  right := FALSE;
  name := '                       ';
  t :=LENGTH(name2);
  FOR i := 1 TO t DO
     name[i] := name2[i];
  i := 1;
  v := 0;
  FOR i := 1 TO t DO
    BEGIN
      IF (name[i] = ' ') THEN
        v := v+1;
    END;
  IF (v = 0) THEN
      right := TRUE
  ELSE
      BEGIN
        WRITELN('NO SPACES ALLOWED, PLEASE REENTER');
        WRITELN;
      END;
  IF right THEN
    BEGIN
      i := 1;
      REPEAT
        v := ORD(name[i]);
        IF((v<=122)AND(v>=97))OR((v>=65)AND(v<=90))OR(v=42) THEN
          i := i + 1
        ELSE
          BEGIN
            WRITELN('INVALID CHARACTER, PLEASE REENTER');
            right := FALSE;
          END;
      UNTIL (i = t+1) OR NOT right;
    END;
END;

{*********************************************************************
*  CH_CASE;  CONVERTS RECEIVER'S NAME TO UPPERCASE BEFORE COMPARING  *
*            IT TO USERNAMES IN IDFILE.DAT                           *
**********************************************************************}
PROCEDURE ch_case(VAR name2 : st_20);

VAR
  name      : STRING[20];
  tp        : INTEGER;

BEGIN
  name := '                    ';
  i := 1;
  tp := LENGTH(name2);
  REPEAT
    name[i] := UpCase(name2[i]);
    i := i + 1;
  UNTIL i = tp + 1;
  name2 := name;
END;

{********************************************************************
*  CHK_USER;  CHECKS TO SEE IF NAME ENTERED AS RECEIVER IS A VALID  *
*             NAME                                                  *
*********************************************************************}
PROCEDURE chk_user(name2 : st_20);

VAR
   temp_name : STRING[20];

BEGIN
   found := FALSE;
   Assign(idlist,idfile_dat);
   IF (name2[1] = '*') THEN
     BEGIN
      IF (name2[2] = ' ') THEN
```

```
        BEGIN
         found := TRUE;
         broadcast :=TRUE;
        END;
     END;
   lock;
   RESET(idlist);
   ctr := 1;
   while (ctr in [1..nodes]) and (not found) do
     begin
       seek(idlist,ctr-1);
       read(idlist,an_entry);
       with an_entry do
        begin
          if (name2 = name) then
            begin
             found := true;
             recv_id := id;
            end;
        end;
       ctr := ctr + 1;
     end;
   CLOSE(idlist);
   unlock;
   IF NOT found THEN
     WRITELN('USER NAME NOT FOUND, TRY AGAIN');
   WRITELN;
 END;

{********************************************************************
*  GET_TIME;  GETS THE HOURS AND MINUTES FROM THE USER TERMINAL     *
*********************************************************************}
PROCEDURE get_time(VAR hour, min : INTEGER);
TYPE
  regpack = RECORD
              ax,bx,cx,dx,bp,si,di,ds,es,flags: INTEGER;
            END;
VAR
  regs : regpack;

BEGIN
  WITH regs DO
  BEGIN
    ax   := $2c00;
    MSDos(regs);
    hour := Hi(cx);
    min  := Lo(cx);
  END;
END;

{********************************************************************
*  HOUR_CALC;  USED IN CALCULATING THE HOUR IF THERE IS OVERLAP     *
*              BETWEEN A SESSION AND THE 24TH HOUR                  *
*********************************************************************}
```

```
PROCEDURE hour_calc;
 BEGIN
  IF(hour2 >= 1) THEN
    IF(min1+min2 >= 60) THEN
      hour:=hour2+1-(24-hour1)
    ELSE
      hour:=hour2-(24-hour1)
  ELSE
    hour:=1;
  IF (hour=0) THEN hour:=24;
END;

{********************************************************************
*  MAIN                                                              *
*********************************************************************}
BEGIN
  putx(1,1,pic);
  canceled_message := FALSE;
  right := FALSE;
  found := FALSE;
  ok  := FALSE;
  if (not reply) then
   begin
     mkwin(1,1,59,6,blue,lightgray,1);
     clrscr;

{********************************************************************
* REPEAT UNTIL NAME IS CORRECT AND VALID                             *
*********************************************************************}
    REPEAT
{********************************************************************
*   REPEAT UNTIL VALID RESPONSE                                      *
*********************************************************************}
      REPEAT
        WRITE(messagea);                {see the user name list?}
        READ(Kbd,listval);
        i := ord(listval);
        if (i = 27) then
          begin
            found := true;
            skip := true;
            right := true;
            ok := true;
          end
        else
          begin
            WRITELN;
            IF (listval = 'Y') OR (listval = 'y') THEN  {if yes display it}
              BEGIN
                ok := true;
                assign(idlist,idfile_dat);
                cursoron(false);
                tempstr := string_reader(45,10,64,21,blue,white,red,white,1,idfile_dat);
     cursoron(true);
    END
   ELSE
    BEGIN
     IF (listval = 'N') OR (listval = 'n') THEN
      BEGIN
       ok := TRUE;
       tempstr := 'ESCAPE';  {set user menu value to escape}
      END
     ELSE
      BEGIN
       ClrScr;
       WRITELN('INVALID RESPONSE, TRY AGAIN PLEASE.');
       ok := FALSE;
      END;
    END;
  end;
UNTIL ok;
if (not skip) then
 begin
  to_name :='          ';
  ClrScr;
  if (tempstr = 'ESCAPE') then
   begin
    WRITELN(message1);
    READLN(to_name);    {enter receiver's username}
    t := length(to_name);
    if (t = 0)  then
     begin
      right := true;
      found := true;
      skip := true;
     end;
    if (not skip) then
     begin
      error_chk(to_name); {check input name for invalid chars/spaces}
      IF right THEN
       BEGIN
        ch_case(to_name);       { if no errors, change to uppercase}
        chk_user(to_name);      { is it a valid username?}
       END;
     end;  {end if not skip}
   end  {end if tempstr}
  else
   begin
    right := true;
    found := true;
    t := length(tempstr);
    if (t < 20) then
     begin
      if (tempstr = 'BROADCAST') then
       begin
        to_name := '#         ';
        broadcast := true;
```

```
          end
         else
          begin
           for i := 1 to t do
            to_name[i] := tempstr[i];
          end;
        end
       else
        to_name := tempstr;
      end;
    end; { end skip if }
  UNTIL right AND found;
  rawin;
 end;  { end IF not reply }

{*********************************************************************
* THIS SECTION OF CODE CALCULATES A TIME-STAMP AND CALLS THE         *
* PROCEDURE THAT WILL BRING UP A SCREEN FOR THE SENDER TO TYPE IN    *
* HIS MESSAGE                                                        *
*********************************************************************)
 if (not skip) and (not for_ward) then
  begin
   Textbackground(blue);
   TextColor(white + blink);
   WRITELN(message2);
   from_and_from_name := from + from_name;
   to_and_to_name := to_u + to_name;
   timestamp;
   x1:=2; y1:=6; x2:=79; y2:=18; back:=blue; fore:=yellow;
   fback:=cyan; ffore:=yellow; edt_file:=in_fil_str;
   bor:=2; outpt:=1;
   editor;
   clrScr;
  end;  {end if}
 from_and_from_name := from + from_name;
 to_and_to_name := to_u + to_name;
END;  { end procedure }
```

APPENDIX I

PROGRAM LISTING

of

GET.INC

```
{**************************************************************************
*  GET.INC  (PREVIOUSLY READFILE.INC)                                     *
*                                                                         *
*  This include file can be called into any window. It creates            *
*  a window of the specified color and location. The window is            *
*  removed when you exit.  This procedure is used to view, save,          *
*  and scroll through messages that have been sent by some user.          *
*                                                                         *
*  The following parameters must be set up before the procedure call:     *
*                                                                         *
*                  x1,        - left column of window  ( >1 )             *
*                  y1,        - top line of window ( >1 )                 *
*                  x2,        - right column of window ( <80)             *
*                  y2,        - bottom line of window ( <25 )             *
*                  back,      - background color of window                *
*                  fore,      - color of text in window                   *
*                  fback,     - bgd color of border (-1 = no border)      *
*                  ffore   : integer;  - clr of line in border            *
*                  edt_file: FilStr;   - name of file to be edited        *
*                  Bor     : integer); - lines in border (0,1,or 2)       *
***************************************************************************}

TYPE
  filstr = STRING[66];     {file string variable}

{**************************************************************************
*  WINDOW_READER;  THIS IS THE MAIN PROCEDURE. ALL THE EDITING FUNCTIONS  *
*                  ARE INCLUDED IN THE PROCEDURE                          *
***************************************************************************}

PROCEDURE window_reader;

{**************************************************************************
*  WINDOW_CORNERS;  DEFINES CORNERS OF MESSAGE DISPLAY WINDOW             *
***************************************************************************}
PROCEDURE window_corners(VAR x1,y1,x2,y2: INTEGER);

VAR
  upper_left_x  : Byte Absolute DSeg:$0004;
  upper_left_y  : Byte Absolute DSeg:$0005;
  lower_right_x : Byte Absolute CSeg:$16a;
  lower_right_y : Byte Absolute CSeg:$16b;

BEGIN
  x1 := upper_left_x + 1;
  y1 := upper_left_y;
  x2 := lower_right_x;
  y2 := lower_right_y;
END;

{**************************************************************************
*  PRE_POPUP_ROUTINES;    PREPARE THE POPUP WINDOWS                       *
***************************************************************************}
```

```
PROCEDURE pre_popup_routines;

VAR
    pg_wnd,
    stk,
    posnum,
    hlp:      INTEGER;
    st :      STRING[20];

BEGIN
   pg_wnd := x1 + TRUNC(((x2 - x1) - 15)/2) + 1;
   hlp := (x2 - (15 + x1));
   frame(x1-1,y1-1,x2+1,y2+1,(fback*16)+ffore,bor);
   Window(x1,y1,x2,y2);
   TextBackGround(back);
   ClrScr;
   fastwrite(x1+30,y1-2,48,' VIEW MESSAGES ');
   fastwrite(23,19,48,'  F2  -  VIEW NEXT WAITING MESSAGE  ');
   fastwrite(23,20,48,'  F3  -  REPLY TO A MESSAGE         ');
   fastwrite(23,21,48,'  F4  -  FORWARD A MESSAGE          ');
   fastwrite(23,22,48,'  F5  -  VIEW SAVED MESSAGES        ');
   fastwrite(23,23,48,'  F6  -  DELETE SAVED MESSAGE       ');
   fastwrite(23,24,48,'  ESC -  EXIT TO MAIN MENU          ');
   Window(x1+1,y1,x2,y2);
END;


{****************************************************************************
*  INITIALIZE;  INITIALIZES KEY VARIABLES IN PREPARATION FOR READING AND    *
*               DISPLAYING MESSAGES                                          *
*****************************************************************************}
PROCEDURE initialize;

BEGIN
   string_length := (x2 - x1) - 1;
   max_lines     := (y2 - y1 + 1);
   right_margin  := string_length + 1;
   bottom_row    := max_lines;
   column        := left_margin;  {current y position for GOTOXY}
   row           := top_row;      {current x position for GOTOXY}
   node_ln       := 1;  {beginning page_buffer array subscript value.   }
   colour        := (back*16)+fore;
END;


{****************************************************************************
*  CUR_UP;  MOVES CURSOR UP ONE LINE.                                        *
*****************************************************************************}
PROCEDURE cur_up;

BEGIN    {user wants to move up a line}
  IF (node_ln - row) > 0 THEN
```

```
    BEGIN
      node_ln := node_ln - row;
      curnd := cur_node(node_ln);
      row := 1;
      GotoXY(1,row);
      InsLine;
      GotoXY(1,row);
      WRITE(curnd^.txt)
    END
END;

{****************************************************************************
*  CUR_DOWN;  MOVES CURSOR DOWN ONE LINE                                    *
****************************************************************************}
PROCEDURE cur_down;

VAR
  nld : INTEGER;

BEGIN
  IF node_ln <= lst^.LENGTH THEN
    BEGIN
      nld := node_ln + (bottom_row - row) + 1;
      IF nld <= lst^.LENGTH THEN
        BEGIN
          node_ln := nld;
          row := bottom_row;
          curnd := cur_node(node_ln);
          GotoXY(1,1);
          DelLine;
          GotoXY(1,bottom_row);
          WRITE(curnd^.txt)
        END
    END
END;

{****************************************************************************
*  CMD_PROCESSOR_2;  KEYBOARD INPUT BUFFER READER                           *
****************************************************************************}
PROCEDURE cmd_processor_2;

VAR
  ord_char : INTEGER; {used to store numeric value assigned to keybrd entry}
  next_fkey : BOOLEAN; {flag set to true when valid function key is entered}

BEGIN
  next_fkey := FALSE;
  REPEAT  {until Next_Fkey}
     READ(Kbd,ch);          {standard keyboard does not echo in Turbo.  }
     ord_char := ORD(ch);   {get the ordinal value of the character read}
     IF (ord_char = escape) THEN
         IF NOT KeyPressed THEN
             BEGIN
               next_fkey := TRUE;
               vchoice  := ord_char;
             END
         ELSE
             BEGIN  {ord_char = Escape; look for function key}
               READ(Kbd,ch);
               ord_char := ORD(ch);
               CASE ord_char OF

                 up         :  cur_up;
                 down       :  cur_down;
                 pgdn       :  page_down;        {display_next_page}
                 pgup       :  page_up;          {display_previous_page}
                 home       :  go_to_top;
                 endkey     :  go_to_end;
                 f2..f6     :  BEGIN
                                 next_fkey := TRUE;
                                 vchoice := ord_char;
                               END;
               END;
             END;   {if escape character}
    GotoXY(column,row);
  UNTIL next_fkey;  {end of the repeat statement}
END;

{**************************************************************************
*  NUM_OF_MSGS;  DETERMINES HOW MANY MESSAGES ARE IN THE NEW OR SAVED     *
*                QUEUE AND SPECIFIES THE POSITION OPEN FOR THE NEXT MSG   *
*                TO BE ADDED TO THE QUEUE                                 *
***************************************************************************}

PROCEDURE num_of_msgs(sub_str : str_66 ; extension : u_exten);

VAR
   temp_num : u_exten;   {string variable to concatenate file number}
   ok       : BOOLEAN;   {flag set to false when open position is found}
   i_num    : INTEGER;   {counter for number of messages}

BEGIN
  i_num := 0;
  ok := TRUE;
    WHILE ok  DO
       BEGIN
         i_num := i_num + 1;
         STR (i_num, temp_num);
         getfile := sub_str + extension + temp_num;
         IF NOT exist(getfile)  THEN
             BEGIN
               ok := FALSE;
               IF  extension = '.N'  THEN  n_add_pos := i_num
               ELSE o_add_pos := i_num;
             END;
       END;
END;
```

```
{*****************************************************************************
*  REORDER_MSG;   INVOKED AFTER A MESSAGE HAS BEEN DELETED.  IT REORDERS    *
*                 THE QUEUE MAINTAINING A FIFO ORDER I.E. SHIFTS MESSAGES   *
*                 DOWN TO FILL AN OPEN POSITION CREATED BY DELETE           *
******************************************************************************}
PROCEDURE reorder_msg (sub_str : str_66 ; extension : u_exten);

VAR
    temp_fil    : str_66;   {file string variable}
    temp_num    : u_exten;  {string variable to concatenate file number}
    cur_pos,                {these integer variables are numeric values}
    cur_hold,               {used as pointers in message queues}
    add_pos,
    add_hold,
    pos_shifts,
    mov_pos     : INTEGER;

  BEGIN
    IF  extension = '.N'  THEN
        BEGIN
          cur_pos := n_cur_pos;
          add_pos := n_add_pos;
        END
    ELSE
        BEGIN
          cur_pos := o_cur_pos;
          add_pos := o_add_pos;
        END;
    pos_shifts := add_pos - cur_pos - 1;
    IF  add_pos = 1  THEN  cur_pos := 0
    ELSE
        BEGIN
          IF  pos_shifts = 0  THEN add_pos := cur_pos
          ELSE
               BEGIN
                 cur_hold := cur_pos;
                 WHILE  pos_shifts <> 0  DO
                     BEGIN
                       mov_pos := cur_pos + 1;
                       STR (mov_pos, temp_num);
                       temp_fil := sub_str + extension + temp_num;
                       Assign (old_fil, temp_fil);
                       STR (cur_pos, temp_num);
                       temp_fil := sub_str + extension + temp_num;
                       Rename (old_fil, temp_fil);
                       cur_pos := cur_pos + 1;
                       add_hold := cur_pos;
                       pos_shifts := pos_shifts - 1;
                     END;     {While Pos_shifts <> 0 }
                 cur_pos := cur_hold;
                 add_pos := add_hold;
               END;
        END;
    IF  extension = '.N'  THEN
```

```
        BEGIN
          n_cur_pos := cur_pos;
          n_add_pos := add_pos;
        END
    ELSE
        BEGIN
          o_cur_pos := cur_pos;
          o_add_pos := add_pos;
        END;
  END;

{***************************************************************************
*  REFRESHER;  REINITIALIZES VARIABLES, READS DESIRED MESSAGE FILE,        *
*              DISPLAYS THE MESSAGE AND INVOKES KEYBOARD BUFFER READER     *
*              UNTIL A KEY IS PRESSED                                      *
***************************************************************************}
PROCEDURE refresher (same_fil : str_66);

BEGIN
  MARK(heaptop);                {save top of heap}
  initialize;
  read_input_file (same_fil);
  IF  first_time  THEN
      BEGIN
        pre_popup_routines;
        first_time := FALSE;
      END;
  display_text (1);
  cmd_processor_2;
  RELEASE(heaptop);             {now put it back}
END;

{***************************************************************************
*  FUNC_ENTRY;  CENTRAL PROCESSING PROCEDURE FOR THE GET INCLUDE FILE.     *
*               DIRECTS PROCESSING BASE ON FUNCTION KEY INPUTS             *
***************************************************************************}
PROCEDURE func_entry;

VAR
     i,
     scount, mcount   : INTEGER; {both used for message number display}
     user_filetable   : ARRAY[1..nodes] OF str_66;
     done,                       {used to terminate function key processor}
     no_messages      : BOOLEAN; {indicates whether there are any new msgs}

     getfile,

     sub_fil_str      : str_66;  {file string variable}
     unum_ext,                   {user number string variable}
     fnum_ext         : u_exten; {file number string variable}

     alright, did     : boolean;
     tfile            : text;
     a_line           : string[80];
```

```
BEGIN
  mcount := 1;
  n_cur_pos := 1;
  o_cur_pos := 0;
  STR (send_id, unum_ext);
  sub_fil_str := usr_path + unum_ext;
  done := FALSE;
  TextBackGround(black);
  ClrScr;
  vchoice := f2;
  did := false;

  REPEAT
    fastwrite(0,3,0,'                                                                         ');
    CASE vchoice OF

{**************************************************************************
*  F2;  INVOKES THE PROCESS WHICH ATTEMPTS TO LOCATE AND DISPLAY THE NEXT *
*       SEQUENTIAL NEW MESSAGE ON FILE                                    *
***************************************************************************}
    f2: BEGIN
          num_of_msgs (sub_fil_str, new_ext);
          IF n_add_pos = 1  THEN
            BEGIN
              getfile := sub_fil_str + dum_ext;
              mcount := 1;
              if (not did) then
                fastwrite(0,3,15,' There are no WAITING messages.                                   ')
              else
                fastwrite(0,3,15,' There are no more WAITING messages.          (previous message saved)');
              no_messages := TRUE;
              refresher(getfile);
              did := false;
            END
          ELSE
            BEGIN
              did := true;
              no_messages := FALSE;
              STR (n_cur_pos, fnum_ext);
              getfile := sub_fil_str + new_ext + fnum_ext;
              fastwrite(0,3,15,'   MESSAGE #');
              Window(1,1,80,25);
              TextColor(white);
              TextBackGround(black);
              GotoXY(13,4);
              if (mcount = 1) then
                WRITE(mcount)
              else
                WRITE(mcount,'       (previous message automatically saved)');
              TextColor(yellow);
              TextBackGround(blue);
              Window(x1+1,y1,x2,y2);
              mcount := mcount + 1;
              refresher (getfile);
              mark(heaptop);
              initialize;
              read_input_file (getfile);
              release(heaptop);
              num_of_msgs (sub_fil_str, new_ext);
              num_of_msgs (sub_fil_str, old_ext);
              Assign (old_fil, getfile);
              STR (o_add_pos, fnum_ext);
              getfile := sub_fil_str + old_ext + fnum_ext;
              Rename (old_fil, getfile);
              reorder_msg (sub_fil_str, new_ext);
              o_add_pos := o_add_pos + 1;
            END;
           if no_messages then
             begin
               REPEAT
                IF (vchoice = f6)or(vchoice = f4)or(vchoice = f3) THEN
                  BEGIN
                   fastwrite(0,3,15,' Invalid selection sequence, try again.                               ');
                   cmd_processor_2;
                   fastwrite(0,3,0,'                                                                       ');
                   delay(200);
                  END;
              UNTIL (vchoice<>f6) and (vchoice<>f3) and (vchoice<>f4);
             end
           else
             begin
               REPEAT
                if (vchoice = f6) then
                  begin
                   fastwrite(0,3,15,' Invalid selection sequence, try again.                               ');
                   cmd_processor_2;
                   fastwrite(0,3,0,'                                                                     ');
                   delay(200);
                  end;
               UNTIL vchoice <> f6;
             end;
        END;

(**************************************************************************
*  F3;  INVOKES THE PROCESS WHICH LETS THE USER REPLY TO A SPECIFIC       *
*       MAIL MESSAGE.                                                     *
***************************************************************************)
    f3: BEGIN
          assign(junk,getfile);
          reset(junk);
          readln(junk,n_line);
          ctr := 1;
          to_name := '                    ';
          for i := 8 to 27 do
             begin
               to_name[ctr] := n_line[i];
               ctr := ctr + 1;
             end;
          close(junk);
          done := true;
          reply := true;
        END;

{###########################################################################
#  F4;  INVOKES THE PROCESS WHICH PERMITS THE USER TO FORWARD A MAIL       #
#       MESSAGE TO ANOTHER USER.                                           #
############################################################################}
    f4: BEGIN
          assign(tfile,getfile);
          reset(tfile);
          assign(infile,in_fil_str);
          rewrite(infile);
          while not eof(tfile) do
            begin
              readln(tfile,a_line);
              writeln(infile,a_line);
            end;
          close(tfile);
          close(infile);
          done := true;
          for_ward := true;
        END;

{###########################################################################
#  F5;  INVOKES THE PROCESS WHICH ATTEMPTS TO LOCATE AND DISPLAY THE NEXT #
#       SEQUENTIAL SAVED MESSAGE                                           #
############################################################################}
    f5: BEGIN
          num_of_msgs (sub_fil_str, old_ext);
          IF o_add_pos = 1  THEN
            BEGIN
              getfile := sub_fil_str + dum_ext;
              fastwrite(0,3,15,' There are no SAVED messages.                                            ');
              refresher (getfile);
              o_cur_pos := 0;
            END
          ELSE
            BEGIN
              if (firstime) then
                begin
                  firstime := false;
                  scount := 1;
                  o_cur_pos := 1;
                  fnum_ext := '1';
                  getfile := sub_fil_str + old_ext + fnum_ext;
                  fastwrite(0,3,15,'   SAVED MESSAGE #');
                  Window(1,1,80,25);
                  TextColor(white);
                  TextBackGround(black);
                  GotoXY(19,4);
                  WRITE(scount);
                  TextColor(yellow);
                  TextBackGround(blue);
                  Window(x1+1,y1,x2,y2);
                  refresher (getfile);
                end
              else
                begin
                  o_cur_pos := o_cur_pos + 1;
                  IF o_cur_pos < o_add_pos  THEN
                    BEGIN
                      scount := scount + 1;
                      STR (o_cur_pos, fnum_ext);
                      getfile := sub_fil_str + old_ext + fnum_ext;
                      fastwrite(0,3,15,'   SAVED MESSAGE #');
                      Window(1,1,80,25);
                      TextColor(white);
                      TextBackGround(black);
                      GotoXY(19,4);
                      WRITE(scount);
                      TextColor(yellow);
                      TextBackGround(blue);
                      Window(x1+1,y1,x2,y2);
                      refresher (getfile);
                    END
                  ELSE
                    BEGIN
                      getfile := sub_fil_str + dum_ext;
                      scount := 0;
                      fastwrite(0,3,15,' There are no more SAVED messages.                               ');
                      refresher (getfile);
                      o_cur_pos := 0;
                    END;
                end;
            END;
          alright := false;
          REPEAT
            IF (vchoice<>f5)and(o_cur_pos=0)and(vchoice<>f2)and(vchoice<>escape) THEN
              BEGIN
                fastwrite(0,3,15,' Invalid selection sequence, try again.                          ');
                cmd_processor_2;
                fastwrite(0,3,0,'                                                                   ');
                delay(200);
              END
            else
              alright := true;
          UNTIL alright;
       END;

{#########################################################################
#  F6;  INVOKES THE PROCESS TO DELETE THE SAVED MESSAGE CURRENTLY        #
#       INDICATED BY O_CUR_POS                                           #
##########################################################################}
    f6: BEGIN
          IF o_cur_pos > 0  THEN
```

```
            BEGIN
              scount := scount - 1;
              num_of_msgs (sub_fil_str, old_ext);
              STR (o_cur_pos, fnum_ext);
              getfile := sub_fil_str + old_ext + fnum_ext;
              Assign (old_fil, getfile);
              Erase (old_fil);
              reorder_msg (sub_fil_str, old_ext);
              IF  o_cur_pos > 0  THEN  o_cur_pos := o_cur_pos -1;
              fastwrite(0,3,15,' Message has been deleted!                                             ');
              getfile := sub_fil_str + dum_ext;
              refresher (getfile);
            END
          ELSE
            BEGIN
              fastwrite(0,3,15,' There are no current SAVED messages to delete!                        ');
              refresher (getfile);
              o_cur_pos := 0;
            END;

          REPEAT
            IF  (vchoice = f3)  OR  (vchoice = f4)  THEN
              BEGIN
                fastwrite(0,3,15,' Invalid selection sequence, try again.                                ');
                cmd_processor_2;
                fastwrite(0,3,0,'                                                                         ');
                delay(200);
               END;
          UNTIL (vchoice <> f3)  AND (vchoice <> f4);

          REPEAT
            IF (vchoice = f6) THEN
              BEGIN
                fastwrite(0,3,15,' You are not reading a message, press F5 to continue.                  ');
                cmd_processor_2;
                fastwrite(0,3,0,'                                                                         ');
                delay(200);
              END;
          UNTIL vchoice <> f6;

    END;

{*************************************************************************
*  ESCAPE;  TERMINATES FUNC_ENTRY PROCEDURE AND CONTROL IS RETURNED TO    *
*           EMAIL PROGRAM                                                 *
**************************************************************************}
    escape: BEGIN
              done := TRUE;
            END;
    END;             {overall Case}
  UNTIL done;
END;
```

```
{***************************************************************************
*  MAIN;  MAIN BODY OF THE WINDOW_READER PROCEDURE                          *
****************************************************************************}
BEGIN
  cursoron(FALSE);
  firstime := true;
  func_entry;
  cursoron(TRUE);
END;                 {procedure Window_Reader}
```

APPENDIX J

PROGRAM LISTING

of

MENU.INC

```
{********************************************************************
*  MENU.INC (PREVIOUSLY USE_MENU)       Bill Saints                 *
*                                                                   *
*   (modified by Gary McAlum for EMAIL, July 1987                   *
*                                                                   *
*  Displays a menu by calling up a file created by MENU_MAKER.COM.*
*  When an item is selected by pressing ENTER, the number of the    *
*  the item is returned. The calling program should include this   *
*  file and make the following declarations:                        *
*                                                                   *
*     Var                                                           *
*       var1 : WinlstHdr;                                           *
*       var2 : integer;                                             *
*                                                                   *
*  The menu file should be read at the beginning of the calling     *
*  program using the following statement:                           *
*                                                                   *
*          var1 := Read_Menu_File(menu_filename)                    *
*                                                                   *
*  The menu is then displayed using the following statement:        *
*                                                                   *
*   var2 := Use_Menu(var1,bor_type,bor_bgclr,bor_fgclr,false);      *
*                                                                   *
*   - bor_type sets the border style:                               *
*          0=solid,1=single line,2=double line, 10=hatched          *
*   - bor_bgclr and bor_fgclr are the border colors                 *
*   - boolean parameter...true=menu stays on screen after choice    *
*                         false=menu disappears after choice        *
*                                                                   *
*  var2 is the number of the menu item that was chosen when the     *
*  user pressed ENTER.                                              *
*                                                                   *
*********************************************************************}

{********************************************************************
*  NODE_POS;                                                        *
*********************************************************************}
FUNCTION node_pos(thislist: winlsthdr; Ptr : winnode): INTEGER;

VAR
   i : INTEGER;
   nd : winnode;

BEGIN
  i := 1;
  nd := thislist^.first;
  WHILE (nd <> Ptr) DO
    BEGIN
      i := i + 1;
      nd := nd^.next;
    END;
  node_pos := i;
END;
```

```
{*******************************************************************
*  CRT_WNLST;  CREATES HEADER FOR LINKED LIST FOR INPUT WINDOWS    *
*******************************************************************}

FUNCTION crt_wnlst: winlsthdr;

VAR
   thishead: winlsthdr;

BEGIN
  NEW(thishead);
  thishead^.LENGTH := 0;
  thishead^.first  := NIL;
  thishead^.last   := NIL;
  crt_wnlst        := thishead;
END;

{******************************************************************
*  INST_WNODE;                                                    *
******************************************************************}
FUNCTION inst_wnode(dat: textstr; x1, x2, lin: INTEGER;
                    prev, nxt: winnode): winnode;

VAR
   thisone: winnode;

BEGIN
  NEW(thisone);
  thisone^.STR    := COPY(dat,1,LENGTH(dat));
  thisone^.prior  := prev;
  thisone^.next   := nxt;
  thisone^.ex1    := x1;
  thisone^.ex2    := x2;
  thisone^.fldlen := (x2 - x1) + 1;
  thisone^.lyne   := lin;
  inst_wnode      := thisone;
END;

{*******************************************************************
* APP_WLST;  APPENDS A NODE ONTO LINKED LIST LST                   *
*******************************************************************}
PROCEDURE app_wlst(thislist: winlsthdr; dat: textstr;
                  x1, x2, lin: INTEGER);

VAR
   thisone: winnode;

BEGIN
  IF thislist^.first = NIL THEN
    BEGIN
      thisone         := inst_wnode(dat,x1,x2,lin,NIL,NIL);
      thislist^.last  := thisone;
      thislist^.first := thisone;
    END
  ELSE
    BEGIN
      thisone   := inst_wnode(dat,x1,x2,lin,thislist^.last,NIL);
      thislist^.last^.next := thisone;
      thislist^.last       := thisone;
    END;
  thislist^.LENGTH := thislist^.LENGTH + 1;
END;


{******************************************************************
*  DEALLOC_MLIST;                                                 *
*******************************************************************}
PROCEDURE dealloc_mlist(wnst: winlsthdr);

VAR
  nextode,
  ode       : winnode;

BEGIN
  if wnst^.first <> nil then
    begin
      ode := wnst^.first;
      WHILE ode <> NIL DO
         BEGIN
           nextode := ode^.next;
           dispose(ode);
           ode := nextode;
         END;
      dispose(wnst);
    end;
END;

{*******************************************************************
*  READ_MENU_FILE;   READS TEXT FROM FILE INTO LINKED LIST.        *
********************************************************************}
FUNCTION read_menu_file(user_file: file_len_66): winlsthdr;

VAR
   len,
   len1,
   len2,
   mchr,
   bx1,bx2,h,i,j,k : INTEGER;
   filvar          : TEXT;     {read from a text file}
   file_exists     : BOOLEAN;
   filstring       : textstr;    {string of 80 chars}
   colour          : STRING[2];
   wlst,
   menlst          : winlsthdr;

{******************************************************************
*  FIND_WINDOW;                                                   *
*******************************************************************}
```

```
PROCEDURE find_window;

VAR
  x1,x2,i,j,k,
  lny,
  chx        : INTEGER;
  ndstr      : textstr;
  fndst,
  done,
  leftbrac   : BOOLEAN;
  ch         : CHAR;
  curnd      : winnode;

BEGIN
  leftbrac := FALSE;
  done := FALSE;
  curnd := menlst^.first;
  lny := 1;

  REPEAT
    chx := 1;
    k := 0;
    i := 0;
    DELETE(ndstr,1,80);
    FOR chx := chx TO LENGTH(curnd^.STR) DO
      BEGIN
        IF leftbrac THEN
          BEGIN
            i := i + 1;
            INSERT(curnd^.STR[chx], ndstr, i);
          END;

        IF curnd^.STR[chx] = '[' THEN
          BEGIN
            x1 := chx;
            leftbrac := TRUE;
            k := 0;
          END;

        IF leftbrac THEN
          IF curnd^.STR[chx] = ']' THEN
            BEGIN
              leftbrac := FALSE;
              i := 0;
              x2 := chx;
              app_wlst(wlst,'',x1,x2,lny);

              wlst^.last^.STR := COPY(ndstr,1,LENGTH(ndstr)-1);
              DELETE(ndstr,1,80);
            END;
        END;
    DELETE(ndstr,1,80);
    curnd := curnd^.next;
    lny := lny + 1;
    IF lny > menlst^.LENGTH THEN
      done := TRUE;
  UNTIL done;
END;

{*****************************************************************
*  READ_INPUT_FILE;                                              *
*****************************************************************}

BEGIN
  menlst := crt_wnlst;
  wlst := crt_wnlst;
  wlst^.menchr := crt_wnlst;
  Assign(filvar,user_file);
  file_exists := exist(user_file);
  IF (file_exists) THEN
    BEGIN
      RESET(filvar);    {get the text file ready to read}
      FOR i := 1 TO 4 DO READLN(filvar, wlst^.box[i]);
      READLN(filvar, wlst^.bgclr);
      READLN(filvar, wlst^.fgclr);
      bx2 := wlst^.box[2] - 1;
      bx1 := wlst^.box[1] - 1;
      len := wlst^.box[3] - bx1;
      FOR i := 1 TO bx2 DO app_wlst(menlst,'',1,5,6);
      FOR i := wlst^.box[2] TO wlst^.box[4] DO
        BEGIN
          READLN(filvar,filstring);
          app_wlst(menlst,filstring,1,5,6);
          FOR k := 1 TO len DO
            BEGIN
             IF filstring[k] = '[' THEN
                filstring[k] := ' ';
             IF filstring[k] = ']' THEN
                  filstring[k] := ' ';
            END;
          app_wlst(wlst^.menchr,filstring,1,5,6);
          FOR j := 1 TO bx1 DO
            INSERT(' ',menlst^.last^.STR,j);
        END;
      CLOSE(filvar);
    END {if exists, INPUT FILE cannot be READ IF it doesn't exist}
  ELSE
    BEGIN
      GotoXY(5,10);
      WRITE('   ****** FILE ', user_file,'  DOES NOT EXIST! ******');
      HALT;
    END;
  find_window;
  wlst^.saviten := wlst^.first;
  read_menu_file := wlst;
  dealloc_mlist(menlst);
END;
```

```
{*******************************************************************
*  MAIN FUNCTION;                                                  *
*******************************************************************}
FUNCTION use_menu(wlstptr: winlsthdr;
                  bordtype,borbg,borfg:INTEGER;
                  stay: BOOLEAN): INTEGER;

CONST
     yes     = 1;
     no      = 0;
     f10     = 68;
     enter   = 13;
     uarr    = 72;
     darr    = 80;
     larr    = 75;
     rarr    = 77;
     home    = 71;
     endkey  = 79;


{*******************************************************************
*  ONE_LINE_INPUT;                                                 *
*******************************************************************}
FUNCTION one_line_input(wlst : winlsthdr;VAR wnode: winnode;
                             limit : INTEGER;
                             VAR ret:INTEGER): INTEGER;

VAR
   i,
   cmd,
   cur,
   index: INTEGER;
   ch    : CHAR;
   done : BOOLEAN;
   in_str : textstr;

BEGIN
  cur := 1;
  in_str := wnode^.STR;
  GotoXY(cur,1);
  done := FALSE;
  ch := ' ';
  cmd := 0;
  WHILE (done <> TRUE) DO
    BEGIN
      READ(Kbd, ch);
      IF ch = CHR(27) THEN
        BEGIN
          if not keypressed then
            begin
              done := true;
              ret := escape;
            end
         else
```

```
            begin
              READ(Kbd, ch);
              cmd := ORD(ch);
              CASE cmd OF
                uarr,darr,larr,rarr : done := TRUE;
               END;
              ret := cmd;
              ch := CHR(0);
            end;
        END
    ELSE
      BEGIN
         IF (ch = CHR(13)) THEN
           BEGIN
             done := TRUE;
             ret := enter;
           END;
      END;
  END;
  IF (NOT stay) OR (ret <> enter) THEN
    BEGIN
      GotoXY(1,1);
      TextBackGround(wlst^.bgclr);
      ClrEol;
      TextColor(wlst^.fgclr);
      GotoXY(2,1);
      WRITE(in_str);
    END;
  one_line_input := node_pos(wlst,wnode);
END;


{********************************************************************
*  PROCEED;                                                         *
*********************************************************************}
PROCEDURE proceed(wlst: winlsthdr; VAR wnode: winnode);

BEGIN
  IF wnode^.next <> NIL THEN
    wnode := wnode^.next
  ELSE
    wnode := wlst^.first;
  click;
END;

{********************************************************************
*  BACKWARD;                                                        *
*********************************************************************}

PROCEDURE backward(wlst: winlsthdr; VAR wnode: winnode);

BEGIN
  IF wnode^.prior <> NIL THEN
    wnode := wnode^.prior
  ELSE
    wnode := wlst^.last;
  click;
END;

{*******************************************************************
*  WRITE_FORM;                                                     *
********************************************************************}
PROCEDURE write_form(wlst: winlsthdr; _color,framecolor: INTEGER);

VAR
   curnd: winnode;
   bx1,bx2,bx3,bx4,
   i,j,lyn : INTEGER;

BEGIN
  Window(wlst^.box[1],wlst^.box[2],wlst^.box[3],wlst^.box[4]);
  ClrScr;
  Window(1,1,80,25);
  bx1 := wlst^.box[1]-1;
  bx2 := wlst^.box[2]-1;
  bx3 := wlst^.box[3]-1;
  bx4 := wlst^.box[4]-1;
  curnd := wlst^.menchr^.first;
  FOR lyn := bx2 TO bx4 DO
    BEGIN
      fastwrite(bx1,lyn,_color,curnd^.STR);
      curnd := curnd^.next;
    END;
  frame(wlst^.box[1],wlst^.box[2],wlst^.box[3],wlst^.box[4],
       framecolor,bordtype);
END;

{*******************************************************************
* INIT;                                                            *
********************************************************************}

PROCEDURE init(wlst: winlsthdr; VAR _color,
              framecolor,retcode: INTEGER);

BEGIN
  framecolor := (borbg*16)+borfg;
  _color := (wlst^.bgclr*16)+wlst^.fgclr;
  TextBackGround(wlst^.bgclr);
  TextColor(wlst^.fgclr);
  retcode := 0;
END;

{******************************************************************
*  CMD_PRO;                                                       *
*******************************************************************}
VAR
  sav_menu : rayptr;
```

```
FUNCTION cmd_pro: INTEGER;

VAR
  fgclr,
  retcode,
  lny,
  x1,x2,
  color,
  framecolor,
  choice    : INTEGER;
  wlst      : winlsthdr;
  wnode     : winnode;

BEGIN
  wlst := wlstptr;
  init(wlst,color,framecolor,retcode);
  sav_menu := getx(wlst^.box[1],wlst^.box[2],wlst^.box[3],wlst^.box[4]);
  write_form(wlst,color,framecolor);
  wnode := wlst^.savitem;

  WHILE (retcode <> enter) and (retcode <> escape) DO
    BEGIN
      x1 := wnode^.ex1;
      x2 := wnode^.ex2;
      lny := wnode^.lyne;
      Window(x1,lny,x2,lny+1);
      IF wlst^.fgclr = yellow THEN
        fgclr := lightgray
      ELSE
        fgclr := wlst^.fgclr;
      TextBackGround(wlst^.fgclr);
      TextColor(wlst^.bgclr);
      GotoXY(1,1);
      ClrEol;
      GotoXY(2,1);
      WRITE (wnode^.STR);
      choice := one_line_input(wlst,wnode, wnode^.fldlen,retcode);
      CASE retcode OF
        uarr,larr: backward(wlst,wnode);
        darr,rarr: proceed(wlst,wnode);
        escape   : choice := 3;
      END;
   END;
  cmd_pro := choice;
  retcode := 200;
  IF stay THEN
    wlst^.savitem := wnode
  ELSE
    begin
     xfree := true;
     putx(wlst^.box[1],wlst^.box[2],sav_menu);
     xfree := false;
    end;
END;
```

```
{*******************************************************************
*  MAIN;                                                           *
*******************************************************************}
BEGIN
  use_menu := cmd_pro;
  Window(1,1,80,25);
END;
```

APPENDIX K

PROGRAM LISTING

of

SEND.INC

```
{**********************************************************************
*  SEND.INC (EDITOR.INC PREVIOUSLY)                                   *
*      Bill Saints                                                    *
*                                                                     *
*   (modified by Gary McAlum for EMAIL, July 1987                     *
*                                                                     *
*  This callable editor can be called into any window. It creates     *
*  a window of the specified color and location. The window is        *
*  removed when you exit from the editor. The editor contains a       *
*  help screen that lists the cursor-control and editing commands.    *
*                                                                     *
*  The following parameters must be set prior to the call:            *
*                                                                     *
*                  x1,        - left column of window  ( >1 )         *
*                  y1,        - top line of window ( >1 )             *
*                  x2,        - right column of window ( <80)         *
*                  y2,        - bottom line of window ( <25 )         *
*                  back,      - background color of window            *
*                  fore,      - color of text in window               *
*                  fback,     - bgd color of border (-1 = no border)  *
*                  ffore   : integer,  - color of line in border      *
*                  edt_file: St_66,    - name of file to be edited    *
*                  Bor     : integer,  - * of border lines (1,2)      *
*                  outpt   : integer); - 1=rewrite file, 2=append     *
*                                                                     *
*  The procedure RE_RITE_TXT(filename) can be called after exiting    *
*  the editor to write the edited text to a file. The procedure       *
*  APP_RITE_TXT can be called to append the edited text to a file     *
*                                                                     *
**********************************************************************}


{**********************************************************************
* Editor; Contains all the editing functions, linked list rountines,  *
*          and all global variables are local to the editor.          *
**********************************************************************}

{**********************************************************************
*  INITIALIZE;                                                        *
**********************************************************************}
PROCEDURE initialize;

BEGIN
   rowset := y1 - 1;
   rowset1 := rowset-1;
   colset  :=x1;
   colset1 := x1+1;
   colour := (back*16)+fore;
   hlpcolor := (lightgray*16)+blue;
   string_length := (x2 - x1) - 1;
   max_lines := (y2 - y1 + 1);
   right_margin := string_length + 1;
   bottom_row := max_lines;
   column       := left_margin;  {current y position for GOTOXY}
   row          := top_row;      {current x position for GOTOXY}
   insertmd     := TRUE;         {user has not pressed INS key.    }
   cnt_flag := 0;
   node_ln := 1;  {beginning page_buffer array subscript value.   }
   FOR i := 1 TO string_length DO
     INSERT(' ',blankstr, 1);
   DELETE(blankstr,string_length,81-string_length);
END;  {procedure initialize}

{*********************************************************************
*  CUR_NODE;  RETURNS A PTR TO THE CURRENT NODE OF LINKED LIST       *
**********************************************************************}
FUNCTION cur_node(POS: INTEGER): nodeptr;

VAR
   i:  INTEGER;
   nd: nodeptr;

BEGIN
  nd := lst^.first;
  FOR i := 2 TO POS DO
    nd := nd^.next;
  cur_node := nd;
END;


{*********************************************************************
*  CRT_DBL;  CREATES HEADER FOR LINKED LIST FOR TEXT LINES           *
**********************************************************************}
FUNCTION crt_dbl: list;

VAR
   thishead: list;

BEGIN
  NEW(thishead);
  thishead^.LENGTH := 0;
  thishead^.first := NIL;
  thishead^.last := NIL;
  crt_dbl := thishead;
END;

{********************************************************************
*  INST_NODE;  CREATES A NEW NODE FOR LINKED LIST                   *
*********************************************************************}
FUNCTION inst_node(dat: txstr; prev, nxt: nodeptr): nodeptr;

VAR
   thisone: nodeptr;

BEGIN
  NEW(thisone);
  thisone^.txt := COPY(dat,1,LENGTH(dat));
  thisone^.prior := prev;
  thisone^.next := nxt;
  thisone^.newln := 1;
  inst_node := thisone;
END;
{*********************************************************************
* APP_LST;  APPENDS A NODE ONTO LINKED LIST                          *
**********************************************************************}

PROCEDURE app_lst(dat: txstr);

VAR
   thisone: nodeptr;

BEGIN
  IF lst^.first = NIL THEN
    BEGIN
      thisone := inst_node(dat,NIL,NIL);
      lst^.last := thisone;
      lst^.first := thisone;
    END
  ELSE
    BEGIN
      thisone := inst_node(dat,lst^.last,NIL);
      lst^.last^.next := thisone;
      lst^.last := thisone;
    END;
  lst^.LENGTH := lst^.LENGTH + 1;
END;

{*********************************************************************
*  INSERT_LST;  INSERTS A NODE INTO THE LINKED LIST                  *
**********************************************************************}
PROCEDURE insert_lst(dat: txstr; POS: INTEGER);

VAR
   temp,
   thisone: nodeptr;
   i: INTEGER;

BEGIN
  IF lst^.LENGTH = 0 THEN
    BEGIN
      thisone := inst_node(dat,NIL,NIL);
      lst^.first := thisone;
      lst^.last := thisone;
    END
  ELSE
    BEGIN
      temp := cur_node(POS);
      IF temp <> NIL THEN
        BEGIN
          thisone := inst_node(dat,temp^.prior,temp);
          IF thisone^.prior <> NIL THEN
```

```
            thisone^.prior^.next := thisone
          ELSE
            lst^.first := thisone;
          temp^.prior := thisone;
        END
      ELSE
        BEGIN
          thisone := inst_node(dat,lst^.last,NIL);
          lst^.last^.next := thisone;
          lst^.last := thisone;
        END;
    END;
  lst^.LENGTH := lst^.LENGTH + 1;
END;

{**********************************************************************
*  DEL_HERE;  DELETES A NODE FROM THE TEXT LINKED LIST AND           *
*             RETURNS THE TEXT STRING FROM THAT NODE                 *
***********************************************************************}
FUNCTION del_here(POS: INTEGER): txstr;

VAR
   temp: nodeptr;

BEGIN
  temp := lst^.first;
  IF POS = 1 THEN
    BEGIN
      lst^.first := temp^.next;
      IF lst^.first <> NIL THEN
        lst^.first^.prior := NIL;
    END
  ELSE
    BEGIN
      temp := cur_node(POS);
      temp^.prior^.next := temp^.next;
      IF temp^.next = NIL THEN
        lst^.last := temp^.prior
      ELSE
        temp^.next^.prior := temp^.prior;
    END;
  del_here := temp^.txt;
  dispose(temp);
  lst^.LENGTH := lst^.LENGTH - 1;
END;

{**********************************************************************
*  APP_RITE_TXT;  APPENDS EDITED TEXT IN ACTIVE WINDOW TO FILE       *
***********************************************************************}
PROCEDURE app_rite_txt(outfile: str_66);

VAR
   filvar:  TEXT;
```

```
BEGIN
  Assign(filvar, outfile);
  IF exist(outfile) THEN
    append(filvar)
  ELSE
    REWRITE(filvar);
  WHILE(lst^.LENGTH <> 0) DO
    WRITELN(filvar, del_here(1));
  CLOSE(filvar);
  dispose(Lst);
END;

{*********************************************************************
*  RE_RITE_TXT;  WRITES TEXT FROM ACTIVE WINDOW INTO FILE.           *
*                INCLUDES THE SENDER'S AND RECEIVER'S USER NAMES AS  *
*                WELL AS WHEN THE MESSAGE WAS SENT                   *
**********************************************************************}

PROCEDURE re_rite_txt(txfile: str_66);

VAR
   filvar:  TEXT;

BEGIN
  Assign(filvar, txfile);
  REWRITE(filvar);
  WRITELN(filvar,from_and_from_name,'TIME MESSAGESENT: ',chour[1],chour[2],':',cmin[1],cmin[2]);
  WRITELN(filvar, to_and_to_name);
  WRITELN(filvar);
  WHILE (lst^.LENGTH <> 0 ) DO
    WRITELN(filvar, del_here(1));
  CLOSE(filvar);
  dispose(Lst);
END;

{********************************************************************
*  WINDOW_CORNERS;                                                  *
*********************************************************************}

PROCEDURE window_corners(VAR x1,y1,x2,y2: INTEGER);

VAR
  upper_left_x  : Byte Absolute DSeg:$0004;
  upper_left_y  : Byte Absolute DSeg:$0005;
  lower_right_x : Byte Absolute CSeg:$16a;
  lower_right_y : Byte Absolute CSeg:$16b;

BEGIN
  x1 := upper_left_x + 1;
  y1 := upper_left_y + 1;
  x2 := lower_right_x;
  y2 := lower_right_y;
END;
```

```
{*********************************************************************
*  FASTWRITE.INC                                                    *
*    Prints a string much faster than writeln.                      *
*                                                                   *
*        The colors are controlled by passing appropriate           *
*      integers in the attrib parameter:                            *
*        Values from 0 to 15 will give you a black background       *
*      and the foreground color associated with the usual           *
*      numbers. From 16 to 31 will give you a blue back-            *
*      ground with the foreground colors cycling through            *
*      again. And so on through 127. For example, passing           *
*      14 in attrib will give you black background and              *
*      yellow foreground. The value 30 will give you blue           *
*      background and yellow foreground.                            *
*        The numbers from 128 through 155 will repeat the           *
*      above sequence but will produce blinking text.               *
*        An easy way to calculate the number you need for           *
*      a particular combination of background and foreground        *
*      colors is to use the folowing formula, substituting          *
*      numbers given in the Turbo Manual for the colors:            *
*                                                                   *
*          (background-color * 16) + foreground-color               *
*                                                                   *
*********************************************************************}
TYPE
   string80 = STRING[80];


PROCEDURE fastwrite(col,row,attrib:Byte;STR:string80);
BEGIN
  InLine
    ($1e/$1e/$8a/$86/row/$b3/$50/$f6/$e3/$2b/$db/$8a/$9e/col/
     $03/$c3/$03/$c0/$8b/$f8/$be/$00/$00/$8a/$be/attrib/
     $8a/$8e/STR/$22/$c9/$74/$3e/$2b/$c0/$8e/$d8/$a0/$49/$04/
     $1f/$2c/$07/$74/$22/$ba/$00/$b8/$8e/$da/$ba/$da/$03/$46/
     $8a/$9a/STR/$ec/$a8/$01/$75/$fb/$fa/$ec/$a8/$01/$74/$fb/
     $89/$1d/$47/$47/$e2/$ea/$2a/$c0/$74/$10/$ba/$00/$b0/
     $8e/$da/$46/$8a/$9a/STR/$89/$1d/$47/$47/$e2/$f5/$1f);
END;


PROCEDURE frame(up_left_x,up_left_y,lo_right_x,lo_right_y,colr,
                 border:INTEGER);

VAR
    x1,y1,x2,y2,
    ul,ur,ll,lr,vert,horz,
    i:             INTEGER;

BEGIN
  y1 := up_left_y - 1;
  y2 := lo_right_y - 1;
  x1 := up_left_x - 1;
  x2 := lo_right_x - 1;
  CASE border OF
    0:BEGIN ul:= 219;ur:=219;ll:=219;lr:=219;vert:=219;horz:=219;END;
    1:BEGIN ul:= 218;ur:=191;ll:=192;lr:=217;vert:=179;horz:=196;END;
    2:BEGIN ul:= 201;ur:=187;ll:=200:lr:=188;vert:=186;horz:=205;END;
    3:BEGIN ul:= 213;ur:=184;ll:=212;lr:=190;vert:=179;horz:=205;END;
    4:BEGIN ul:= 214;ur:=183;ll:=211;lr:=189;vert:=186;horz:=196;END;
    5:BEGIN ul:= 204;ur:=185;ll:=204;lr:=185;vert:=186;horz:=205;END;
   10:BEGIN ul:= 176;ur:=176;ll:=176;lr:=176;vert:=176;horz:=176;END;
   11:BEGIN ul:= 177;ur:=177;ll:=177;lr:=177;vert:=177;horz:=177;END;
   12:BEGIN ul:= 178;ur:=178;ll:=178;lr:=178;vert:=178;horz:=178;END;
  END;

  fastwrite(x1,y1,colr,CHR(ul));
  FOR i := x1 + 1 TO x2 - 1 DO
    fastwrite(i,y1,colr,CHR(horz));
  fastwrite(i+1,y1,colr,CHR(ur));
  FOR i := y1 + 1 TO y2 - 1 DO
    BEGIN
      fastwrite(x1,i,colr,CHR(vert));
      fastwrite(x2,i,colr,CHR(vert));
    END;

  fastwrite(x1,y2,colr,CHR(ll));
  FOR i := x1 + 1 TO x2 - 1 DO
    fastwrite(i,y2,colr,CHR(horz));
  fastwrite(i+1,y2,colr,CHR(lr));
END;


{*************************************************************************
* WAITKEY; PAUSES PROGRAM FOR KEYBOARD INPUT. YES PARAMETER CAUSES *
*          PROMPT MESSAGE TO BE SHOWN WHEREVER CURSOR IS LOCATED.   *
*************************************************************************}

PROCEDURE waitkey(prompt: INTEGER);

VAR
   c : CHAR;

BEGIN
  IF prompt = yes THEN
    WRITE('** PRESS SPACEBAR TO CONTINUE **');
  REPEAT
    READ(Kbd,c);
  UNTIL ORD(c) = 32;
END;


{*************************************************************************
* READ_INPUT_FILE; READS TEXT FROM EXISTING FILE INTO LINKED LIST. *
*                  IF FILE DOESN'T EXIST, JUST CREATE ONE NODE      *
*                  WITH BLANK IN IT.                                *
*************************************************************************}
```

```
PROCEDURE read_input_file(user_file: str_66);

VAR
   i,len           : INTEGER;
   commentfile     : TEXT;     {read from a text file}
   file_exists     : BOOLEAN;
   instr           : txstr;  {string of 80 chars}

BEGIN
  Lst := crt_dbl;
  Assign(commentfile,user_file);
  file_exists := exist(user_file);
  IF (file_exists) THEN
    BEGIN
      RESET(commentfile);    {get the text file ready to read}
      WHILE NOT EOF(commentfile) DO
        BEGIN
          READLN(commentfile,instr);
          len:=LENGTH(instr);
          app_lst(instr);
          DELETE (lst^.last^.txt,string_length,81-string_length);
        END;  {while}
     lst^.last^.newln := 1;
     CLOSE(commentfile);
    END {if exists}  {INPUT FILE cannot be READ IF it doesn't exist}
  ELSE
    insert_lst('    ', 1);
END;

{************************************************************************
*  PRE_POPUP_ROUTINES;                                                  *
*************************************************************************}
PROCEDURE pre_popup_routines;
CONST     {prepare the popup windows. }
  exit_message = '  F10 = SEND  ';
  help_win_msg = ' F1 = HELP ';
  cancel_msg   = ' ALT F9 = CANCEL ';
VAR
    pg_wnd,
    stk,
    hlp:      INTEGER;
BEGIN
   pg_wnd := x1 + TRUNC(((x2 - x1) - 15)/2) + 1;
   hlp := (x2 - (15 + x1));
   frame(x1-1,y1-1,x2+1,y2+1,(fback*16)+ffore,bor);
   fastwrite(x1+2,y2,ffore,exit_message);
   fastwrite(hlp-1,y2,ffore,help_win_msg);
   fastwrite(x1+2,y1-2,ffore,cancel_msg);
   Window(x1,y1,x2,y2);
   TextBackGround(back);
   ClrScr;
   Window(x1+1,y1,x2,y2);
END;
```

```
{**********************************************************************
*  DISPLAY_WARNING;  TOP AND END OF FILE WARNINGS                     *
***********************************************************************}
PROCEDURE display_warning(hey_there:  string12);

VAR x,y :INTEGER;

BEGIN
   x:=WhereX;
   y:=WhereY;
   mkwin(x1+1,y1,x1+20,y1+3,black,green,3);
   GotoXY(3,2);
   WRITELN(bell,hey_there);  {ring the bell and display warning msg.}
   Delay(500);   {milliseconds.}
   rmwin;
   Window(x1+1,y1,x2,y2);
   GotoXY(x,y);
END;

{**********************************************************************
*  DISPLAY_TEXT;  DISPLAYS A PAGE OF TEXT STARTING WITH NODE_NUM      *
***********************************************************************}
PROCEDURE display_text(node_num: INTEGER);

VAR
     x_1, x_2, y_1, y_2,rw,col,end_line, start_line, i: INTEGER;
     nd: nodeptr;

BEGIN
  end_line    := (max_lines);
  start_line  := (end_line - (max_lines - 1));
  col         := left_margin;
  window_corners(x_1,rw,x_2,y_2);
  x_1         := x_1 - 1;
  ClrScr;                   {clear off any old text.}
  {display the appropriate text on the screen.}
  nd := cur_node(node_num);
  rw := rw - 1;
  FOR i := start_line TO end_line DO
    BEGIN
      IF nd <> NIL THEN
        BEGIN
          fastwrite(x_1,rw,colour,nd^.txt);
          nd := nd^.next;
        END;
      rw := rw + 1;           {increment for the next row.}
    END; {for}
  curnd :=cur_node(node_ln);
  GotoXY(column,row);
END;

{**********************************************************************
*  SHO_MODE;  DISPLAYS INSERT/OVERWRITE MODE ABOVE WINDOW.            *
***********************************************************************}
```

```
PROCEDURE sho_mode;

 VAR   loc: INTEGER;
BEGIN
  loc := (x2 - (15 + x1));
  Window(x1,y1-1,x2,y1+1);
  GotoXY(loc,1);
  TextBackGround(black);
  TextColor(ffore);
  IF insertmd = TRUE THEN
    WRITE(' Insert Mode  ')
  ELSE
    WRITE('Overwrite Mode');
  Window(x1+1,y1,x2,y2);
  TextBackGround(back);
  TextColor(fore);
  GotoXY(column,row);
END;

{*********************************************************************
*  INS_MODE;  Sets insert/overwrite mode.                            *
**********************************************************************}
PROCEDURE ins_mode;

BEGIN
  insertmd := NOT insertmd;
  sho_mode;
END;

{*********************************************************************
*  DEL_REST_LINE;  DELETES THE LINE FROM THE CURSOR ON.              *
**********************************************************************}
PROCEDURE del_rest_line;

BEGIN
  DELETE(curnd^.txt,column, LENGTH(curnd^.txt) - column + 1);
  ClrEol;
END;

{*********************************************************************
*  TAB_TO_BLANK;  MOVES CURSOR TO NEXT BLANK TO THE RIGHT.           *
**********************************************************************}
PROCEDURE tab_to_blank;

BEGIN
  IF column < LENGTH(curnd^.txt) THEN
    BEGIN
      column :=column + 1;
      WHILE (curnd^.txt[column] <> ' ') DO
        column := column + 1;
      IF column > LENGTH(curnd^.txt) THEN
        column := column - 1;
    END;
END;
```

```
{*********************************************************************
*  TAB_LEFT;  MOVES CURSOR TO FIRST BLANK TO THE LEFT.               *
*********************************************************************}
PROCEDURE tab_left;

BEGIN
  IF column > 1 THEN
    BEGIN
      column := column - 1;
      WHILE (curnd^.txt[column] <> ' ') AND (column <> 1) DO
        column := column - 1;
    END;
END;

{*********************************************************************
*  PAGE_DOWN;  DISPLAYS NEXT SCREEN PAGE                             *
*********************************************************************}
PROCEDURE page_down;

VAR
   node_num: INTEGER;

BEGIN
  IF node_ln < lst^.LENGTH THEN
    BEGIN
      IF ((node_ln + max_lines) <= lst^.LENGTH) THEN
        BEGIN
          node_ln := node_ln + max_lines;
          column := 1;
          node_num := node_ln - row + 1;
        END
      ELSE
        BEGIN
          column := 1;
          IF lst^.LENGTH >= max_lines THEN
            BEGIN
              IF (max_lines - row) < (lst^.LENGTH - node_ln) THEN
                BEGIN
                  node_num := node_ln + (max_lines - row) +1;
                  IF((lst^.LENGTH-node_ln)-(max_lines-row)) < row THEN
                    BEGIN
                      row := (lst^.LENGTH - node_ln)-(max_lines-row);
                      node_ln := lst^.LENGTH;
                    END
                 ELSE
                   BEGIN
                    node_ln := node_ln + max_lines;
                  END;
                END
              ELSE
                BEGIN
                  node_num := lst^.LENGTH - row + 1;
                  node_ln := lst^.LENGTH;
                END;
            END
          ELSE
            BEGIN
              node_num := lst^.LENGTH - row + 1;
              node_ln := lst^.LENGTH;
            END;
        END;
      display_text(node_num);
    END
  ELSE
    display_warning(end_of_file);
END;

{**********************************************************************
*  PAGE_UP;  DISPLAYS PREVIOUS SCREEN PAGE                            *
**********************************************************************}
PROCEDURE page_up;

VAR
   node_num: INTEGER;

BEGIN
  IF node_ln > 1 THEN
    BEGIN
      IF ((node_ln - max_lines) > 0) THEN
        BEGIN
          node_ln := node_ln - max_lines;
          column := 1;
          IF node_ln < row THEN
            row := node_ln;
          node_num := node_ln - row + 1
        END
      ELSE
        BEGIN
          column := 1;
          row := 1;
          node_ln := 1;
          node_num := 1;
        END;
      display_text(node_num);
    END
  ELSE
    display_warning(top_of_file);
END;

{**********************************************************************
*  GO_TO_TOP;  MOVES CURSOR TO FIRST LINE IN FILE                     *
**********************************************************************}
PROCEDURE go_to_top;

BEGIN
  column := 1;
  row := 1;
  node_ln := 1;
  curnd := lst^.first;
  display_text(1);
END;


{*************************************************************************
*  GO_TO_END;  MOVES CURSOR TO LAST LINE OF FILE                         *
*************************************************************************}
PROCEDURE go_to_end;
VAR
    node_num: INTEGER;

BEGIN
  node_ln := lst^.LENGTH;
  column := 1;
  node_num := (lst^.LENGTH - max_lines) + 1;
  curnd := lst^.last;
  IF node_num < 1 THEN
    BEGIN
      node_num := 1;
      row := lst^.LENGTH;
    END
  ELSE
    row := bottom_row;
    display_text(node_num);
END;

{*************************************************************************
*  CUR_LEFT;  MOVES CURSOR ONE CHAR SPACE TO THE LEFT                    *
*************************************************************************}
PROCEDURE cur_left;

BEGIN  {left arrow}
  IF (column > left_margin) THEN
    column := column - 1
  ELSE
    IF node_ln > 1 THEN
      BEGIN
        node_ln := node_ln - 1;
        curnd := curnd^.prior;
        column := (LENGTH(curnd^.txt) + 1);
        IF row > top_row THEN
          row := row - 1
        ELSE
          BEGIN
            GotoXY(1,row);
            InsLine;
            GotoXY(1,row);
            WRITE(curnd^.txt);
          END;
      END
    ELSE
      display_warning(top_of_file);
END;
```

```
{**********************************************************************
*  CUR_RIGHT;  MOVES CURSOR ONE SPACE TO RIGHT.                       *
***********************************************************************}
PROCEDURE cur_right;

BEGIN
  IF (column < right_margin) THEN
   BEGIN
     IF (column <= (LENGTH(curnd^.txt))) THEN
      column := column + 1
     ELSE
       IF node_ln <> lst^.LENGTH THEN
         BEGIN
           column := 1;
           node_ln := node_ln + 1;
           curnd := curnd^.next;
           IF row < bottom_row THEN
             row := row + 1
           ELSE
             BEGIN
               GotoXY(1,top_row);
               DelLine;
               GotoXY(1,row);
               WRITE(curnd^.txt);
             END;
         END;
     END
  ELSE
    display_warning(end_of_file);
END;

{**********************************************************************
*  CUR_UP;  MOVES CURSOR UP ONE LINE.                                 *
***********************************************************************}
PROCEDURE cur_up;

BEGIN    {user wants to move up a line}
  IF (row > top_row) THEN
    BEGIN
      row := row - 1;
      node_ln := node_ln - 1;
      curnd := curnd^.prior;
      IF column > LENGTH(curnd^.txt) THEN
         column := (LENGTH(curnd^.txt) + 1);
    END
  ELSE
    IF node_ln > 1 THEN
      BEGIN
        GotoXY(1,row);
        InsLine;
        GotoXY(1,row);
        curnd := curnd^.prior;
        WRITE(curnd^.txt);
        node_ln := node_ln - 1;
      END
    ELSE
      display_warning(top_of_file);
END;

(********************************************************************
*  CUR_DOWN;  MOVES CURSOR DOWN ONE LINE                             *
*********************************************************************)
PROCEDURE cur_down;

BEGIN
  IF node_ln < lst^.LENGTH THEN
    BEGIN
      IF (row < bottom_row) THEN
        BEGIN
          row := row + 1;
          node_ln := node_ln + 1;
          curnd := curnd^.next;
          IF column > LENGTH(curnd^.txt) THEN
             column := LENGTH(curnd^.txt) + 1;
        END
      ELSE
        BEGIN
          GotoXY(1,1);
          DelLine;
          GotoXY(1,bottom_row);
          curnd := curnd^.next;
          WRITE(curnd^.txt);
          node_ln := node_ln + 1;
          IF column > LENGTH(curnd^.txt) THEN
             column := LENGTH(curnd^.txt) + 1;
        END;
    END
  ELSE
    display_warning(end_of_file);
END;

(*********************************************************************
*  DEL_LINE;  DELETES THE LINE ON WHICH THE CURSOR IS LOCATED.        *
**********************************************************************)
PROCEDURE del_line;

VAR
    st:  txstr;
    nd:  nodeptr;

BEGIN
  IF curnd^.next <> NIL THEN
    BEGIN
      curnd := curnd^.next;
      st := del_here(node_ln);
      GotoXY(1,row);
      DelLine;
      IF ((bottom_row - row) <= (lst^.LENGTH - node_ln)) THEN
```

```
        BEGIN
          GotoXY(1,bottom_row);
          nd := cur_node(node_ln + (bottom_row - row));
          WRITE(nd^.txt);
        END;
    END
  ELSE
    BEGIN
      IF lst^.LENGTH > 1 THEN
        BEGIN
          curnd := curnd^.prior;
          curnd^.newln := 1;
          st := del_here(node_ln);
          node_ln := node_ln - 1;
          GotoXY(1,row);
          ClrEol;
          IF row > 1 THEN
            row := row - 1
          ELSE
            BEGIN
              GotoXY(1,1);
              WRITE(curnd^.txt);
            END;
        END
      ELSE
        BEGIN
          DELETE(curnd^.txt,1,string_length);
          GotoXY(1,1);
          ClrEol;
        END;
    END;
  column := 1;
END;

{*********************************************************************
*  PUT_TOGETHER; CONCATENATES STRING WITH CURRENT NODE STRING        *
**********************************************************************}
FUNCTION put_together(str1: string160; tx: txstr): string160;

VAR
    len:    INTEGER;

BEGIN
  len := LENGTH(str1);
  len := len + 1;
  INSERT(tx,str1,len);
  IF str1[len-1] <> ' ' THEN
    INSERT(' ',str1,len);
  put_together := str1;
END;

{*********************************************************************
*  CHOP; CUTS THE STRING TO SIZE AND RETURNS THE REMAINDER.          *
**********************************************************************}
```

```
FUNCTION chop(VAR str1: string160): string160;

VAR
    i,j,
    count,
    len:    INTEGER;
    str2:   string160;

BEGIN
  i := (string_length);      {find out how much needs to be wrapped}
  IF column <= i THEN
    j := 0
  ELSE
    j := 1;
  IF LENGTH(str1) >= i THEN
    BEGIN
      REPEAT
        i := i - 1;
        IF (i < column) THEN
          j := j + 1;
      UNTIL (str1[i] = ' ') OR (i = 1);
      count := LENGTH(str1) - i;
      IF i > 1 THEN
        BEGIN
          str2 := COPY(str1,(i+1),count);
          cnt_flag := cnt_flag + 1;
        END
      ELSE
        BEGIN
          i := string_length;
          str2 := COPY(str1,i,1);
          IF column >= string_length THEN
            BEGIN
              column := column + 1;
              clf := 2;
            END;
          j := j + 1;
        END;
      DELETE(str1,i,(count + 1));
      IF cnt_flag = 1 THEN
        clf := j;       { count - (string_length - column); }
    END
  ELSE
    BEGIN
      clf := column;
      DELETE(str2,1,160);
    END;
  chop := str2;
END;


{**********************************************************************
*  WRAP; PERFORMS WRAP-AROUND FUNCTION                                 *
**********************************************************************}
```

```
PROCEDURE wrap(rt: INTEGER; str1: string160);

VAR
   len,
   count,
   rw,cl,
   i,j,ndl: INTEGER;
   str2: string160;
   nd: nodeptr;
   done: BOOLEAN;

BEGIN
   nd := curnd;
   ndl := node_ln;
   rw := row;
   cl := column;
   cnt_flag := 0;

      IF (rt = 1) THEN
        str1 := put_together(str1,curnd^.txt)
      ELSE
        str1 := COPY(curnd^.txt,1,LENGTH(curnd^.txt));

      IF (LENGTH(str1) >= string_length) THEN
        str2 := chop(str1)
      ELSE
        DELETE(str2,1,160);

      done := FALSE;
      WHILE (LENGTH(str1) > 0) AND (done = FALSE) DO
        BEGIN
          len := LENGTH(str1);
          IF curnd^.newln = 1 THEN   {a paragraph marker in this line}
            BEGIN
              curnd^.txt := COPY(str1,1,len);
              IF row <= bottom_row THEN
                BEGIN
                  fastwrite(colset,rowset1+row,colour,blankstr);
                  fastwrite(colset,rowset1+row,colour,curnd^.txt);
                END;

              IF LENGTH(str2) > 0 THEN
                BEGIN
                  curnd^.newln := 0;
                  insert_lst(str2,(node_ln + 1));
                  curnd^.next^.newln := 1;
                  IF row < bottom_row THEN
                    BEGIN
                      GotoXY(1,(row+1));
                      InsLine;
                      fastwrite(colset,rowset+row,colour,curnd^.next^.txt);
                    END;
                  done := TRUE;
                END
              ELSE
                DELETE(str1,1,160);
            END
          ELSE               {no paragraph marker in this line}
            BEGIN
              curnd^.txt := COPY(str1,1,len);
              IF row <= bottom_row THEN
                BEGIN
                  GotoXY(1,row);
                  WRITE(curnd^.txt);
                  ClrEol;
                END;
              IF LENGTH(str2) = 0 THEN
                BEGIN
                  done := TRUE;
                END
              ELSE
                BEGIN
                  curnd := curnd^.next;
                  node_ln := node_ln + 1;
                  row := row + 1;
                  str1 := put_together(str2,curnd^.txt);
                  IF LENGTH(str1) >= string_length THEN
                    str2 := chop(str1)
                  ELSE
                    BEGIN
                      done := TRUE;
                      curnd^.txt := COPY(str1,1,LENGTH(str1));
                      IF row <= bottom_row THEN
                        BEGIN
                          GotoXY(1,row);
                          WRITE(curnd^.txt);
                        END;
                    END;
                END;
            END;
        END;

  curnd := nd;
  node_ln := ndl;
  row := rw;
  column := cl;
  cnt_flag := 0;
END;

{*********************************************************************
*  DEL_CH_LEFT;   DELETES CHARACTER TO LEFT OF CURSOR                *
**********************************************************************}
PROCEDURE del_ch_left(dlt: INTEGER);

VAR
    st: txstr;
    temp,
    nd:   nodeptr;
    ndl,
    rw,
    POS,
    len,
    len2: INTEGER;
    upln: BOOLEAN;
    bigstr: string160;

BEGIN
  nd := curnd;
  ndl := node_ln;
  rw := row;
  upln := FALSE;

  IF dlt = 1 THEN
    column := column + 1;
  IF column > left_margin THEN
    column := column - 1
  ELSE
    IF (row = top_row) THEN
      BEGIN
        IF curnd = lst^.first THEN
          exit
        ELSE
          BEGIN                     {top_row but not the first node}
            nd := curnd^.prior;
            nd^.newln := 0;
            IF LENGTH(curnd^.txt) = 0 THEN
              BEGIN
                st := del_here(node_ln);
                lst^.last^.newln := 1;
                GotoXY(1,row);
                DelLine;
              END;
            ndl := node_ln - 1;
            column := LENGTH(nd^.txt) + 1;
            upln := TRUE;
            GotoXY(1,row);
            InsLine;
            WRITE(nd^.txt);
          END;
      END
    ELSE                            {at left margin but not top row}
      BEGIN
        nd := curnd^.prior;
        nd^.newln := 0;
        IF LENGTH(curnd^.txt) = 0 THEN
          BEGIN
            st := del_here(node_ln);
            lst^.last^.newln := 1;
            GotoXY(1,row);
            DelLine;
            IF (bottom_row - row) <= (lst^.LENGTH - node_ln) THEN
              BEGIN
              POS := node_ln + (bottom_row - row);
              GotoXY(1,bottom_row);
              temp := cur_node(POS);
              WRITE(temp^.txt);
            END;
        END;
      ndl := node_ln - 1;
      column := LENGTH(nd^.txt) + 1;
      rw := row - 1;
      upln := TRUE;
    END;

IF upln = FALSE THEN
  BEGIN
    len := LENGTH(curnd^.txt) - column;
    IF len > 0 THEN
      BEGIN
        st := COPY(curnd^.txt,column+1, len);
        DELETE(curnd^.txt,column,len+1);
        INSERT(st,curnd^.txt,column);
        GotoXY(column,row);
        WRITE(st);
        ClrEol;
      END
    ELSE
      BEGIN
        DELETE(curnd^.txt,column,1);
        GotoXY(column,row);
        WRITE(' ');
      END;
  END;

WHILE(curnd^.newln = 0) AND (upln = FALSE) DO
    BEGIN
      DELETE(wrapper,1,160);
      wrapper := COPY(curnd^.txt,1,LENGTH(curnd^.txt));
      len := LENGTH(wrapper)+1;
      INSERT(curnd^.next^.txt,wrapper,len);
      INSERT(' ',wrapper,len);
      bigstr := chop(wrapper);
      len2 := LENGTH(wrapper);
      IF len2 > (len-1) THEN               {if wrap around is needed}
        BEGIN
          GotoXY(1,row);
          WRITE(wrapper);
          curnd^.txt := COPY(wrapper,1,len2);
          IF LENGTH(bigstr) = 0 THEN
            BEGIN
              curnd^.newln := 1;
              st := del_here(node_ln + 1);
              GotoXY(1,row + 1);
              DelLine;
              IF (bottom_row - row) <= (lst^.LENGTH - node_ln) THEN
                BEGIN
```

```
                    POS := node_ln + (bottom_row - row);
                    GotoXY(1,bottom_row);
                    temp := cur_node(POS);
                    WRITE(temp^.txt);
                  END;
              END
            ELSE
              BEGIN
                curnd^.next^.txt := COPY(bigstr,1,LENGTH(bigstr));
                GotoXY(1,row + 1);
                ClrEol;
                GotoXY(1,row+1);
                WRITE(curnd^.next^.txt);
              END;
            curnd := curnd^.next;
            row := row + 1;
            node_ln := node_ln + 1;
          END
        ELSE
          upln := TRUE;
       END;

  curnd := nd;
  node_ln := ndl;
  row := rw;

END;

{#####################################################################
#  ENTER_KEY;  ENDS LINE, INSERTS PARAGRAPH MARKER, AND MOVES CURSOR #
#              AND ANY REMAINING TEXT TO NEXT LINE.                  #
#####################################################################}
PROCEDURE enter_key;

VAR
    len :  INTEGER;

BEGIN
  IF column = 1 THEN              {if cursor is at column #1}
    BEGIN
      insert_lst('',node_ln);     {insert a node in linked list   }
      IF row <> bottom_row THEN   {if cursor is not at bottom row }
        BEGIN
          InsLine;                {insert a blank line in window  }
          row := row + 1;         {increment row number           }
          node_ln := node_ln + 1; {increment node line number     }
          curnd := cur_node(node_ln); {get pointer to current node}
        END
      ELSE
        BEGIN
          InsLine;
          GotoXY(1,1);
          DelLine;
          GotoXY(column,row);
        node_ln := node_ln + 1;
        curnd := cur_node(node_ln);
        fastwrite(colset,rowset1+1,colour,curnd^.txt);
      END;
  END
ELSE
  BEGIN
    len := LENGTH(curnd^.txt);
    IF column <= len THEN    {if cursor is not at column one}
      BEGIN
        wrapper := COPY(curnd^.txt,column, {get remainder of string}
            len - column + 1);
        DELETE(curnd^.txt,column,   {delete remainder of node str  }
            len - column + 1);
        GotoXY(column,row);
        ClrEol;
        IF curnd^.newln = 1 THEN
          BEGIN
            insert_lst(wrapper,node_ln + 1);
            IF row < bottom_row THEN
              BEGIN
                GotoXY(1,row+1);
                InsLine;
                fastwrite(colset,rowset+row,colour,curnd^.next^.txt);
              END;
          END;
        curnd := curnd^.next;
        column := 1;
        IF row < bottom_row THEN
          row := row + 1
        ELSE
          BEGIN
            GotoXY(1,1);
            DelLine;
            GotoXY(1,row);
            IF curnd^.prior^.newln = 1 THEN
              fastwrite(colset,rowset1+row,colour,curnd^.txt);
          END;
        node_ln := node_ln + 1;
        IF curnd^.prior^.newln = 0 THEN
          wrap(1,wrapper);
        curnd^.prior^.newln := 1;
      END
    ELSE                         {cursor is one past end of str  }
      BEGIN
        column := 1;
        insert_lst('',node_ln + 1);
        IF row < bottom_row THEN
          row := row + 1
        ELSE
          BEGIN
            GotoXY(1,1);
            DelLine;
          END;
          GotoXY(column,row);
          InsLine;
          node_ln := node_ln + 1;
          curnd^.newln := 1;
          curnd := curnd^.next
        END;
    END;
END;

{*********************************************************************
*  ENTER_TEXT;  HANDLES THE INPUT OF TEXT CHARACTERS                 *
**********************************************************************}
PROCEDURE enter_text;

VAR
   len: INTEGER;

BEGIN  {text}
  IF insertmd THEN
    INSERT(ch,curnd^.txt,column)
  ELSE
    BEGIN
      INSERT(ch,curnd^.txt,column);
      DELETE(curnd^.txt,column+1,1);
    END;

  IF (LENGTH(curnd^.txt) < string_length) THEN
    BEGIN
      fastwrite(colset,rowset1+row,colour, curnd^.txt);
      column := column + 1;
    END
  ELSE
    BEGIN
      column := column + 1;
      wrap(0,wrapper);
      IF column > (LENGTH(curnd^.txt)+1) THEN
        BEGIN
          curnd := curnd^.next;
          node_ln := node_ln + 1;
          column := clf;
          IF column < 1 THEN
            column := 1;
          IF row <> bottom_row THEN
              row := row + 1
          ELSE
            BEGIN
              GotoXY(1,bottom_row);
              WRITE(curnd^.prior^.txt);
              ClrEol;
              GotoXY(1,1);
              DelLine;
              GotoXY(1,bottom_row);
              WRITE(curnd^.txt);
            END;
        END;
    END;
END;

{***********************************************************************
*  HELP_SCREEN;                                                        *
************************************************************************}
PROCEDURE help_screen;

VAR
  x,y,lne : INTEGER;

BEGIN
  x:=WhereX;
  y:=WhereY;
  mkwin(column,row,column+20,row+5,blue,lightgray,2);
  Window(1,1,80,25);
  cursoron(FALSE);
  ClrScr;
  lne := 1;
  fastwrite(1,lne,hlpcolor,'                  ***  HELP SCREEN FOR TEXT EDITOR  ***');
  lne := lne + 2;
  fastwrite(1,lne,hlpcolor,'     CURSOR CONTROL:');
  lne := lne + 1;
  fastwrite(1,lne,hlpcolor,'     ===============');
  lne := lne + 1;
  fastwrite(1,lne,hlpcolor,'    Arrow keys     - Move one character or line at a time.');
  lne := lne + 1;
  fastwrite(1,lne,hlpcolor,'    <PgUp>, <PgDn> - Move one window page at a time');
  lne := lne + 1;
  fastwrite(1,lne,hlpcolor,'    Tab Key        - Move to next blank space to the right');
  lne := lne + 1;
  fastwrite(1,lne,hlpcolor,'    <Shift>/<Tab>  - Move to next blank space to left');
  lne := lne + 1;
  fastwrite(1,lne,hlpcolor,'    <Ctrl>/->      - Move to end of line');
  lne := lne + 1;
  fastwrite(1,lne,hlpcolor,'    <Ctrl>/<-      - Move to beginning of line');
  lne := lne + 1;
  fastwrite(1,lne,hlpcolor,'    <Home>         - Move to beginning of file');
  lne := lne + 1;
  fastwrite(1,lne,hlpcolor,'    <End>          - Move to end of file');
  lne := lne + 2;
  fastwrite(1,lne,hlpcolor,'    EDITING FUNCTIONS:');
  lne := lne + 1;
  fastwrite(1,lne,hlpcolor,'    =================');
  lne := lne + 1;
  fastwrite(1,lne,hlpcolor,'    <Bksp>         - Delete character to left of cursor');
  lne := lne + 1;
  fastwrite(1,lne,hlpcolor,'    <Del>          - Delete character at cursor');
  lne := lne + 1;
  fastwrite(1,lne,hlpcolor,'    <Ctrl>/<Bksp>  - Delete line');
  lne := lne + 1;
  fastwrite(1,lne,hlpcolor,'    <Ctrl>/<End>   - Delete rest of line');
  lne := lne + 1;
  fastwrite(1,lne,hlpcolor,'    <Enter>        - End paragraph, move cursor to line');
  lne := lne + 1;
  fastwrite(1,lne,hlpcolor,'    <Ins>          - Toggle between insert/ overwrite mode');
  fastwrite(26,23,31,' PRESS SPACEBAR TO RETURN ');
  waitkey(no);
  rmwin;
  Window(x1+1,y1,x2,y2);
  GotoXY(x,y);
  TextBackGround(back);
  cursoron(TRUE);
END;

{**********************************************************************
*  CMD_PROCESSOR;                                                     *
**********************************************************************}
PROCEDURE cmd_processor;

VAR
  ord_char : INTEGER;
  finished : BOOLEAN;

BEGIN
  finished := FALSE;
  REPEAT  {until Finished}
     READ(Kbd,ch);       {standard keyboard does not echo in Turbo.  }
     ord_char := ORD(ch); {get the ordinal value of the character read}
     IF (ord_char <> escape) THEN
       BEGIN
         CASE ord_char OF

           tab       :  tab_to_blank;

           bksp      :  del_ch_left(0);

           enter     :  enter_key;

           ctrl_bksp :  del_line;

           ELSE
             enter_text;        {handles text input from keyboard}
          END;
        END
     ELSE
       BEGIN            {ord_char = Escape; look for function key}
         READ(Kbd,ch);
         ord_char := ORD(ch);
         CASE ord_char OF

           ins       :  ins_mode;

           s_tab     :  tab_left; {tab left to first available space}

           left      :  cur_left;
```

```
            right       :  cur_right;

            up          :  cur_up;

            down        :  cur_down;

            ctrl_left   :  column := left_margin;

            ctrl_right:    column := LENGTH(curnd^.txt) + 1;

            del         :  del_ch_left(1);

            ctrl_end    :  del_rest_line;

            pgdn        :  page_down; {display_next_page}

            pgup        :  page_up;   {display_previous_page}

            home        : go_to_top;

            endkey      : go_to_end;

            f1          :  help_screen;

            alt_f9      :  BEGIN
                             outpt := 0;
                             finished := TRUE;
                             canceled_message :=TRUE;
                             IF exist('D:\mail\in.fil') THEN
                               BEGIN
                                 Assign(erase_fil,'D:\mail\in.fil');
                                 CLOSE(erase_fil);
                                 Erase(erase_fil);
                               END;
                           END;

            f10         :  BEGIN
                             finished := TRUE;
                           END;
            END;           {second nested case}
        END;                 {if escape character}
    GotoXY(column,row);
  UNTIL finished;          {end of the repeat statement}
END;

{*******************************************************************************
*  EDITOR;  THIS IS THE MAIN PROCEDURE. ALL THE EDITING FUNCTIONS              *
*             ARE CONTROLLED BY THE CASE STATEMENTS BELOW.                     *
********************************************************************************}
PROCEDURE editor;

BEGIN    {the main brain for electronic nominal group program}
  initialize;
  read_input_file(edt_file);
  pre_popup_routines;
  display_text(1);
  sho_mode;
  cmd_processor;
  CASE outpt OF
    1: re_rite_txt(edt_file);
    2: app_rite_txt(edt_file);
  END;
END;  {procedure Editor}
```

APPENDIX L

PROGRAM LISTING

of

STRMENU.INC

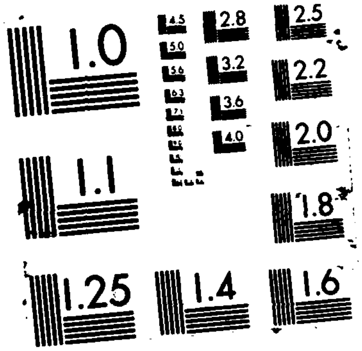
```
{*********************************************************************
*    STRING_READER.INC          Bill Saints                          *
*                                                                    *
*   (modified by Gary McAlum for EMAIL, July 1987                    *
*                                                                    *
*  The String_Reader creates a window with the specified dimensions.*
*  It reads a list of strings from a text file and displays a        *
*  highlighted cursor. When you press ENTER, it returns the string   *
*  from the cursor position. If the ESCAPE key is pressed, the       *
*  string 'ESCAPE' is returned. The window is removed when a         *
*  selection is made.                                                *
*                                                                    *
*  The following parameters must be passed in the call:              *
*                                                                    *
*  String_Reader  (x1,        - left column of window  ( >1 )        *
*                  y1,        - top line of window ( >1 )            *
*                  x2,        - right column of window ( <80)        *
*                  y2,        - bottom line of window ( <25 )        *
*                  back,      - background color of window           *
*                  fore,      - color of text in window              *
*                  fback,     - bgd color of border (-1 = no border)*
*                  ffore,     - clr of line in border                *
*                  Bor     : integer); - lines in border (0,1,or 2) *
*                  edt_file: FilStr;   - name of file                *
*                                                                    *
*  Here is a program that calls the editor into a full screen.       *
*                                                                    *
*    Program Texted(Input,Output);                                   *
*                                                                    *
*    include   STRMENU.INC                                           *
*                                                                    *
*    Begin                                                           *
*      CursorOn(false);                                              *
*      String_Reader(2,2,79,24,blue,white,red,blue,2,'Filename');   *
*      CursorOn(true);                                               *
*    End.                                                            *
*********************************************************************}

Type
  fil_len_66 = String[66];
  LString    = String[72];

Function exst(user_file: fil_len_66):  boolean;

var  fil : file;
Begin
    Assign(fil, user_file);
    {$I-}
    reset(fil);
    {$I+}
    exst := (IOresult = 0);
    close(fil);
End;
```

```
{****************************************************************
*  WINDOW_CORNERS;                                              *
****************************************************************}
Procedure wind_corners(Var x1,y1,x2,y2: integer);

Var
  Upper_Left_X  : Byte Absolute Dseg:$0004;
  Upper_Left_Y  : Byte Absolute Dseg:$0005;
  Lower_Right_X : Byte Absolute Cseg:$16A;
  Lower_Right_Y : Byte Absolute Cseg:$16B;

Begin
  x1 := Upper_Left_X+1;
  y1 := Upper_Left_Y+1;
  x2 := Lower_Right_X;
  y2 := Lower_Right_Y;
End;



{----------------------------------------------------------------}
{ The GET_ATTRIB procedure returns the foreground color attribute  }
{ (0-31) and the background color attribute (0-7) at the requested }
{ cursor location.                                                 }
{ Copyright 1986, 1987 by Shenandoah Software Technologies         }
{----------------------------------------------------------------}
Type
  RegPack     = record
                  ax, bx, cx, dx, bp, si, di, ds, es, flags : integer
                end;

PROCEDURE get_attr(    x, y : integer;
                     var fattr, battr : integer);

var
  regs                                        : regpack;
  savex, savey, blinking, intensity           : integer;

begin
  if not (y in [1..25]) then exit;                { Validate parameters }
  if not (x in [1..80]) then exit;
  savex := wherex;                          { save original cursor location }
  savey := wherey;
  gotoxy(x,y);
  regs.ax := $0f00;
  intr($10,regs);                               { get current page in BH }
  regs.ax := $0800;
  intr($10,regs);                                  { get attribute in AH }
  gotoxy(savex,savey);
  if (regs.ax and $8000) = $8000 then           { set blinking indicator }
    blinking := 1
  else
    blinking := 0;
  if (regs.ax and $0800) = $0800 then          { set intensity indicator }
```

```
    intensity := 1
  else
    intensity := 0;
                            { set foreground and background attribute fields }
  fattr := ((regs.ax and $0700) div 256) + (intensity * 8) +
           (blinking * 16);
  battr := (regs.ax and $7000) div 4096
end;




{------------------------------------------------------------
=  POP_WINDOW; Opens a window, saving the background, and   =
=    returns a pointer to the window information. Parameters=
=    are as follows:                                         =
=      sex1 - X1 (upper left corner)                         =
=      why1 - Y1     "    "    "                             =
=      sex2 - X2 (lower right corner)                        =
=      why2 - Y2    "     "      "                           =
=      back - background color                               =
=      fore - foreground color                               =
=      bbg  - background border color                        =
=      bfg  - foreground border color                        =
=      Bor  - border type:   0 = no border                   =
=                            1 = single line                 =
=                            2 = double line                 =
=                            3 = solid border (fg color)     =
=                            10,11,12 = different hatches    =
------------------------------------------------------------}
Type
  winfo = ^winpop;
  winpop = record
            xx : integer;
            yy : integer;
            x1, y1, x2, y2 : integer;
            Curx1, Cury1, Curx2, Cury2 : integer;
            bg, fg                     : integer;
            csbg,csfg,cbbg,cbfg        : integer;
            restor : rayptr;
            bor    : integer;
            oldx   : integer;
            oldy   : integer;
          end;

{-----------------------------------------------------------
=  POP_WINDOW;                                              =
-----------------------------------------------------------}
Function Pop_Window(sex1,why1,sex2,why2,back,fore,bbg,bfg,Bor : integer): winfo;

Var
  Thiswin : winfo;
  Bgclr,
  Fgclr  : integer;
```

```
Begin
  new(Thiswin);
  Thiswin^.xx := wherex;
  Thiswin^.yy := wherey;
  Thiswin^.bor := Bor;
  Thiswin^.csbg := back;
  Thiswin^.csfg := fore;
  Thiswin^.cbbg := bbg;
  Thiswin^.cbfg := bfg;
  Thiswin^.Oldx := sex1;
  Thiswin^.Oldy := why1;

  wind_corners(Thiswin^.x1,Thiswin^.y1,Thiswin^.x2,Thiswin^.y2);
  get_attr(Thiswin^.xx,Thiswin^.yy,Thiswin^.fg,Thiswin^.bg);
  Thiswin^.restor := getx(sex1,why1,(sex2-sex1)+1,(why2-why1)+1);
  if bor > 0 then
    begin
      Thiswin^.Curx1 := sex1+1;
      Thiswin^.Cury1 := why1+1;
      Thiswin^.Curx2 := sex2-1;
      Thiswin^.Cury2 := why2-1;
      frame(sex1,why1,sex2,why2,(bbg*16)+bfg,Bor);
    end
  else
    begin
      Thiswin^.Curx1 := sex1;
      Thiswin^.Cury1 := why1;
      Thiswin^.Curx2 := sex2;
      Thiswin^.Cury2 := why2;
    end;

  window(Thiswin^.Curx1,Thiswin^.Cury1,Thiswin^.Curx2,Thiswin^.Cury2);
  textbackground(back);
  textcolor(fore);
  clrscr;
  Pop_Window := Thiswin;
End;

{----------------------------------------------------------
=  UNPOP; Removes a window from screen and deallocates the  =
=   window record. Will return control to window that was   =
=   active when this window was opened. After doing Unpop   =
=   to a window, you must use Pop_Window to create it again =
=   before it can be used again.                            =
----------------------------------------------------------}
Procedure Unpop(Thiswin : winfo);

Begin
  xfree := true;
  putx(Thiswin^.Oldx,Thiswin^.Oldy,Thiswin^.restor);
  xfree := false;
  window(Thiswin^.x1,Thiswin^.y1,Thiswin^.x2,Thiswin^.y2);
  textbackground(Thiswin^.bg);
  textcolor(Thiswin^.fg);
  gotoxy(Thiswin^.xx,Thiswin^.yy);
  dispose(Thiswin);
End;

{-----------------------------------------------------------
=  SETCOLOR;                                               =
-----------------------------------------------------------}
Procedure Setcolor(bg,fg: integer);

Begin
  textbackground(bg);
  textcolor(fg);
End;

Type
  St72 = String[72];

function TrimL(InpStr: LString): LString;
{ strip leading spaces from a String }
Var i,len : Integer;
Begin
   len := length(InpStr);
   i := 1;
   While (i <= len) and (InpStr[i] = ' ') do
      i := i + 1;
   TrimL := Copy(InpStr,i,len-i+1)
End;

function TrimR(InpStr: LString): LString;
{ strip trailing spaces from a String }
Var i : Integer;
Begin
   i := length(InpStr);
   While (i >= 1) and (InpStr[i] = ' ') do
      i := i - 1;
   TrimR := Copy(InpStr,1,i)
End;



Type
  FilStr = String[72];

{*************************************************************************
*  String_Reader;  THIS IS THE MAIN PROCEDURE. ALL THE EDITING FUNCTIONS  *
*      ARE INCLUDED IN THE PROCEDURE.                                     *
*************************************************************************}
Function String_Reader(x1, y1, x2, y2,back,fore,fback,ffore: integer;
                                  Bor                     : integer;
                                  edt_file                : FilStr):Filstr;
Const
  { refer to Turbo Pascal version 3.0 appendix K for keyboard return codes}
  F1     = 59;
  F10    = 68;
  Enter  = 13;
  Escape = 27;  {function keys, arrows, etc. use an ascii escape sequence}
  LEFT   = 75;  {left arrow key}
  RIGHT  = 77;  {right arrow key}
  UP     = 72;  {up arrow key}
  DOWN   = 80;  {down arrow key}
  HOME   = 71;  {home key; go to top of file, or first page}
  ENDKEY = 79;  {go to page with last line of text on it. }
  PgUp   = 73; {goto previous page}
  PgDn   = 81; {goto next page}
    left_margin         =  1;  {beginning column of each line.}
    top_row             =  1;  {beginning row of each page starts here.}


Type
  Str2                           = String[2];
  Txstr                          = String[80];

{******* RECORD FOR LINKED LIST NODE ***********}
    NodePtr = ^Node;
    Node = record
           txt     : Txstr;
           next    : NodePtr;
           prior   : NodePtr;
           newln   : integer;
    end;

{******* RECORD FOR LINKED LIST HEADER ***********}
    List = ^Head;
    Head = record
           length: integer;
           first:  NodePtr;
           Last:   NodePtr;
    end;


VAR
   readwin   : Winfo;
   fil       : FilStr;
   curnd     : NodePtr;
   ReadLst       : List;
   user_file : FilStr;
   ch     : char;  {ch = trapped character from keyboard.  }
   stack_counter,          {keep track of open windows for closing.}
   i, j,
   _color,
   Revcolor,
   colset,
   rowset,
   max_lines, bottom_row,  {max_lines = bottom_row.}
   column, row                        : integer;
   string_length,
```

AD-A185 758   OPTIMIZATION ON ENHANCEMENT OF THE PLEXSYS NETWORK   3/3
MESSAGING SYSTEM (EMAIL)(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH G D MCALUM 1987
UNCLASSIFIED   AFIT/CI/NR-87-76T   F/G 25/5   NL


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

```
   right_margin,
   node_ln         : integer;            {number of node in linked list}


{*****************************************************************
*  CUR_NODE;  RETURNS A PTR TO THE CURRENT NODE OF LINKED LIST   *
*****************************************************************}
Function curnode(pos: integer): NodePtr;

Var
   i:  integer;
   nd: NodePtr;

Begin
  nd := ReadLst^.first;
  for i := 2 to pos do
    nd := nd^.next;
  curnode := nd;
End;


{*****************************************************************
*  CRT_DBL;  CREATES HEADER FOR LINKED LIST FOR TEXT LINES       *
*****************************************************************}
Function crtdbl: List;

Var
   thishead: List;

Begin
  new(thishead);
  thishead^.length := 0;
  thishead^.first := nil;
  thishead^.last := nil;
  crtdbl := thishead;
End;

{****************************************************************
*  INST_NODE;  CREATES A NEW NODE FOR LINKED LIST               *
****************************************************************}
Function ins_node(dat: txstr; prev, nxt: NodePtr): NodePtr;

Var
   thisone: NodePtr;

Begin
  new(thisone);
  thisone^.txt := Copy(dat,1,Length(dat));
  thisone^.prior := prev;
  thisone^.next := nxt;
  thisone^.newln := 1;
  ins_node := thisone;
End;
```

```
{****************************************************************
* APP_LST;  APPENDS A NODE ONTO LINKED LIST                    *
****************************************************************}
Procedure ap_list(dat: txstr);

Var
   thisone: NodePtr;

Begin
  if ReadLst^.first = nil then
    begin
      thisone := ins_node(dat,nil,nil);
      ReadLst^.last := thisone;
      ReadLst^.first := thisone;
    end
  else
    begin
      thisone := ins_node(dat,ReadLst^.last,nil);
      ReadLst^.last^.next := thisone;
      ReadLst^.last := thisone;
    end;
  ReadLst^.length := ReadLst^.length + 1;
End;

{***************************************************************************
*  READ_INPUT_FILE;   READS TEXT FROM EXISTING FILE INTO LINKED LIST.      *
*        IF FILE DOESN'T EXIST, JUST CREATE ONE NODE WITH BLANK IN IT.    *
***************************************************************************}
Procedure read_infile(user_file: FilStr);

Var
   instr           : string[20];  {string of 20 chars}

Begin
  ReadLst := crtdbl;
  lock;
  reset(idlist);
  instr := 'BROADCAST';
  ap_list(instr);
  ctr := 1;
  while ctr in [1..nodes] do
    begin
      seek(idlist,ctr-1);
      read(idlist,an_entry);
      with an_entry do
        begin
          if (name <> blanks20) then
            begin
              instr := name;
              ap_list(instr);
            end;
        end;
      ctr := ctr + 1;
    end;
  close(idlist);
  unlock;
End;  {read_infile}

{*************************************************************************
*  PRE_POPUP_ROUTINES;                                                    *
*************************************************************************}
Procedure pre_popup;

Var
  promptx1 : integer;

Begin
  Readwin := Pop_Window(X1-1, Y1-1, X2+1, Y2+1,back,fore,fback, ffore,Bor);
  window(x1,y1,x2,y2);
  promptx1 := x1 + trunc(((x2-x1)-14)/2) - 1;
  fastwrite(promptx1,Y2,(fback*16)+ffore,' ESCAPE = Quit ');
  fastwrite(promptx1-1,Y1-2,(fback*16)+ffore,' ENTER = Choice ');
End;      {procedure pre_popup}

{*****************************************************************************
*  INITIALIZE;                                                                *
*****************************************************************************}
Procedure init;

Var
  tempclr : integer;

Begin
   string_length := (x2 - x1)-1;
   colset := x1-1;
   rowset := y1 - 2;
   max_lines := (Y2 - Y1) + 1;
   right_margin := string_length + 1;
   bottom_row := max_lines;
   column       := left_margin;  {current y position for GOTOXY}
   row          := top_row;     {current x position for GOTOXY}
   node_ln      := 1;  {beginning page_buffer array subscript value.    }
   _color       := (back*16)+fore;
   if fore > 7 then
     begin
       tempclr := fore - 8;
       if tempclr = 6 then
         tempclr := 7;
     end
   else
     tempclr := fore;
   Revcolor     := (tempclr*16)+back
End;  {procedure init}


{******************************************************************************
*  DISPLAY_TEXT;  DISPLAYS A PAGE OF TEXT STARTING WITH NODE # node_num.     *
******************************************************************************}
```

```
Procedure disp_text(node_num: integer);

Var
     x_1, x_2, y_1, y_2,rw,end_line, start_line, i: integer;
     nd: NodePtr;

Begin
  end_line    := (max_lines);
  start_line  := (end_line - (max_lines - 1));

  wind_corners(x_1,rw,x_2,y_2);
  x_1 := pred(x_1);
  rw := pred(rw);
  clrscr;                  {clear off any old text.}
              {display the appropriate text on the screen.}
  nd := curnode(node_num);
  for i := start_line to end_line do
    begin
      if nd <> nil then
        begin
          fastwrite(x_1,rw,_color,nd^.txt);
          nd := nd^.next;
        end;
      rw := rw + 1;           {increment for the next row.}
    end;
  curnd :=curnode(node_ln);
  gotoxy(column,row);
End;


{**************************************************************************
*  PAGE_DOWN;  DISPLAYS NEXT SCREEN PAGE                                  *
**************************************************************************}
Procedure pagedown;

Var
   node_num: integer;

Begin
  if node_ln < ReadLst^.length then
    begin
      if ((node_ln + max_lines) <= ReadLst^.length) then
        begin
          node_ln := node_ln + max_lines;
          node_num := node_ln - row + 1;
        end
      else
        begin
          if ReadLst^.length >= max_lines then
            begin
              if (max_lines - row) < (ReadLst^.length - node_ln) then
                begin
                  node_num := node_ln + (max_lines - row) +1;
                  if ((ReadLst^.length - node_ln) - (max_lines - row)) < row then
```

```
                    begin
                      row := (ReadLst^.length - node_ln) - (max_lines - row);
                      node_ln := ReadLst^.length;
                    end
                 else
                   begin
                    node_ln := node_ln + max_lines;
                  end;
                end
              else
                begin
                  node_num := ReadLst^.length - row + 1;
                  node_ln := ReadLst^.length;
                end;
            end
          else
            begin
              node_num := ReadLst^.length - row + 1;
              node_ln := ReadLst^.length;
            end;
        end;
      disp_text(node_num);
      fastwrite(colset,rowset+row,Revcolor,curnd^.txt);
    end;
End;

{**************************************************************************
*  PAGE_UP;  DISPLAYS PREVIOUS SCREEN PAGE                                *
**************************************************************************}
Procedure pageup;

Var  node_num: integer;

Begin
  if node_ln > 1 then
    begin
      if ((node_ln - max_lines) > 0) then
        begin
          node_ln := node_ln - max_lines;
          column := 1;
          if node_ln < row then
            row := node_ln;
          node_num := node_ln - row + 1
        end
      else
        begin
          row := 1;
          node_ln := 1;
          node_num := 1;
        end;
      disp_text(node_num);
      fastwrite(colset,rowset+row,Revcolor,curnd^.txt);
    end;
End;
```

```
{************************************************************************
*  goto_TOP;  MOVES CURSOR TO FIRST LINE IN FILE                       *
************************************************************************}
Procedure goto_Top;

Begin
  row := 1;
  node_ln := 1;
  curnd := ReadLst^.first;
  disp_text(1);
  fastwrite(colset,rowset+row,Revcolor,curnd^.txt);
End;

{*********************************************************************
*  goto_END;  MOVES CURSOR TO LAST LINE OF FILE                     *
*********************************************************************}
Procedure goto_End;

Var
    node_num: integer;

Begin
  node_ln := ReadLst^.length;
  node_num := (ReadLst^.length - max_lines) + 1;
  curnd := ReadLst^.last;
  if node_num < 1 then
    begin
      node_num := 1;
      row := ReadLst^.length;
    end
  else
    row := bottom_row;
    disp_text(node_num);
  fastwrite(colset,rowset+row,Revcolor,curnd^.txt);
End;


{*****************************************************************
*  curUP;  MOVES CURSOR UP ONE LINE.                             *
*****************************************************************}
Procedure curUp;

Begin    {user wants to move up a line}
  if node_ln > 1 then
    begin
      click;
      fastwrite(colset,rowset+row,_color,curnd^.txt);
      node_ln := node_ln - 1;
      curnd := curnd^.prior;
      if (row > top_row) then
        begin
          row := row - 1;
          fastwrite(colset,rowset+row,Revcolor,curnd^.txt);
        end
      else
        if node_ln > 0 then
          begin
            gotoxy(1,row);
            InsLine;
            fastwrite(colset,rowset+row,Revcolor,curnd^.txt);
          end;
    end;
End;

{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$  curDOWN;  MOVES CURSOR DOWN ONE LINE                        $
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
Procedure curdown;

Begin
  if node_ln < ReadLst^.length then
    begin
      click;
      fastwrite(colset,rowset+row,_color,curnd^.txt);
      node_ln := node_ln + 1;
      curnd := curnd^.next;
      if (row < bottom_row) then
        begin
          row := row + 1;
          fastwrite(colset,rowset+row,Revcolor,curnd^.txt);
        end
      else
        begin
          gotoxy(1,1);
          DelLine;
          fastwrite(colset,rowset+row,Revcolor,curnd^.txt);
        end;
    end;
End;

{-----------------------------------------------------------
=  CMD_PROCESSOR;                                          =
-----------------------------------------------------------}
Procedure cmd_pro;

Var
  ord_char : integer;
  Finished : boolean;
  St       : Txstr;

Begin
  Finished := false;
  fastwrite(colset,rowset+row,Revcolor,curnd^.txt);
  while keypressed do
    read(kbd,ch);
  repeat  {until Finished}
     read(kbd,ch);          {standard keyboard does not echo in Turbo.  }
     ord_char := ord(ch);   {get the ordinal value of the character read}
     if (ord_char <> Escape) then
       begin
         if ord_char = Enter then
           begin
             Unpop(readwin);
             Finished := true;
           end;
       end
     else
       begin
         if keypressed then
           begin  {ord_char = Escape; look for function key}
             read(kbd,ch);
             ord_char := ord(ch);
             case ord_char of
               UP        :  curUp;
               DOWN      :  curdown;
               PgDn      :  pagedown;               { display_next_page; }
               PgUp      :  pageup;                 { display_previous_page; }
               Home      : goto_Top;
               EndKey    : goto_End;
               F10       : begin
                             Unpop(readwin);
                             Finished := true;
                           end;
             end;
           end
         else
           begin
             Unpop(readwin);
             Finished := true;
             curnd^.txt := 'ESCAPE';
           end;
       end;
    gotoxy(column,row);
  until Finished;  {end of the repeat statement}
  St := curnd^.txt;
  St := TrimL(St);
  St := TrimR(St);
  String_Reader := St;
End;

(****************************************************************************)
begin
  init;
  read_infile(edt_file);
  pre_popup;
  disp_text(1);
  cmd_pro;
End;  {function String_Reader}
```

APPENDIX M

PROGRAM LISTING

of

EINIT.BAT

```
echo off
rem  **********************************************************************
rem  *                                                                    *
rem  * EINIT.BAT  -  This batch file will do several things needed to     *
rem  * initialize the network EMAIL system.  The most important task      *
rem  * is to build the file called IDFILE.DAT.  This is done by exe-      *
rem  * cuting BLD_FILE.COM.  This is a user-friendly, menu driven pro-    *
rem  * gram which is self-explanatory.  After building the file, it is    *
rem  * copied to the subdirectory MAIL which is located in each user's    *
rem  * directory.  In addition, all extraneous messages and temp files    *
rem  * are deleted also.  These files are only related to EMAIL and do    *
rem  * affect any other part of the network.  Presently, this batch file  *
rem  * is set up to initialize 16 user directories.  If the network is    *
rem  * expanded to 32 nodes, simply remove the commented code out, which  *
rem  * is pointed out below.  This file also initializes the system time  *
rem  * file by calling GTIME.COM from the SESSMAN directory.               *
rem  *                                                                    *
rem  **********************************************************************

rem  Call bldfile.com to build the initial IDFILE.DAT
cd..
cd public
cls
echo off
if exist temp.fil  ren temp.fil,flag.fil
if exist idfile.dat    del idfile.dat
bldfile
if log == %1  echo ********************
if log == %1  echo * EMAIL log is set *
if log == %1  echo ********************

echo **********************************************************
echo *                                                        *
echo *         INITIALIZING PLEXSYS EMAIL SYSTEM              *
echo *                                                        *
echo **********************************************************
rem   ****** Clean out user directories  *****

echo     Cleaning out user1\mail directory
cd user1
cd mail
copy c:\public\idfile.dat,idfile.dat
if exist temp.fil ren temp.fil, flag.fil
if exist user1.n* del user1.n*
if exist user1.o* del user1.o*
if exist in.fil   del in.fil
if exist *.mes    del *.mes
cd..
cd ..
echo     Cleaning out user2\mail directory
cd user2
cd mail
copy c:\public\idfile.dat,idfile.dat
```

```
if exist temp.fil ren temp.fil, flag.fil
if exist user2.n* del user2.n*
if exist user2.o* del user2.o*
if exist in.fil   del in.fil
if exist *.mes    del *.mes
cd..
cd ..
echo     Cleaning out user3\mail directory
cd user3
cd mail
copy c:\public\idfile.dat,idfile.dat
if exist temp.fil ren temp.fil, flag.fil
if exist user3.n* del user3.n*
if exist user3.o* del user3.o*
if exist in.fil   del in.fil
if exist *.mes    del *.mes
cd..
cd ..
echo     Cleaning out user4\mail directory
cd user4
cd mail
copy c:\public\idfile.dat,idfile.dat
if exist temp.fil ren temp.fil, flag.fil
if exist user4.n* del user4.n*
if exist user4.o* del user4.o*
if exist in.fil   del in.fil
if exist *.mes    del *.mes
cd..
cd ..
echo     Cleaning out user5\mail directory
cd user5
cd mail
copy c:\public\idfile.dat,idfile.dat
if exist temp.fil ren temp.fil, flag.fil
if exist user5.n* del user5.n*
if exist user5.o* del user5.o*
if exist in.fil   del in.fil
if exist *.mes    del *.mes
cd..
cd ..
echo     Cleaning out user6\mail directory
cd user6
cd mail
copy c:\public\idfile.dat,idfile.dat
if exist temp.fil ren temp.fil, flag.fil
if exist user6.n* del user6.n*
if exist user6.o* del user6.o*
if exist in.fil   del in.fil
if exist *.mes    del *.mes
cd..
cd ..
echo     Cleaning out user7\mail directory
cd user7
cd mail
```

```
copy c:\public\idfile.dat,idfile.dat
if exist temp.fil ren temp.fil, flag.fil
if exist user7.n* del user7.n*
if exist user7.o* del user7.o*
if exist in.fil   del in.fil
if exist *.mes    del *.mes
cd..
cd ..
echo     Cleaning out user8\mail directory
cd user8
cd mail
copy c:\public\idfile.dat,idfile.dat
if exist temp.fil ren temp.fil, flag.fil
if exist user8.n* del user8.n*
if exist user8.o* del user8.o*
if exist in.fil   del in.fil
if exist *.mes    del *.mes
cd..
cd ..
echo     Cleaning out user9\mail directory
cd user9
cd mail
copy c:\public\idfile.dat,idfile.dat
if exist temp.fil ren temp.fil, flag.fil
if exist user9.n* del user9.n*
if exist user9.o* del user9.o*
if exist in.fil   del in.fil
if exist *.mes    del *.mes
cd..
cd ..
echo     Cleaning out user10\mail directory
cd user10
cd mail
copy c:\public\idfile.dat,idfile.dat
if exist temp.fil  ren temp.fil, flag.fil
if exist user10.n* del user10.n*
if exist user10.o* del user10.o*
if exist in.fil    del in.fil
if exist *.mes     del *.mes
cd..
cd ..
echo     Cleaning out user11\mail directory
cd user11
cd mail
copy c:\public\idfile.dat,idfile.dat
if exist temp.fil  ren temp.fil, flag.fil
if exist user11.n* del user11.n*
if exist user11.o* del user11.o*
if exist in.fil    del in.fil
if exist *.mes     del *.mes
cd..
cd ..
echo     Cleaning out user12\mail directory
cd user12
cd mail
copy c:\public\idfile.dat,idfile.dat
if exist temp.fil  ren temp.fil, flag.fil
if exist user12.n* del user12.n*
if exist user12.o* del user12.o*
if exist in.fil    del in.fil
if exist *.mes     del *.mes
cd..
cd ..
echo     Cleaning out user13\mail directory
cd user13
cd mail
copy c:\public\idfile.dat,idfile.dat
if exist temp.fil  ren temp.fil, flag.fil
if exist user13.n* del user13.n*
if exist user13.o* del user13.o*
if exist in.fil    del in.fil
if exist *.mes     del *.mes
cd..
cd ..
echo     Cleaning out user14\mail directory
cd user14
cd mail
copy c:\public\idfile.dat,idfile.dat
if exist temp.fil  ren temp.fil, flag.fil
if exist user14.n* del user14.n*
if exist user14.o* del user14.o*
if exist in.fil    del in.fil
if exist *.mes     del *.mes
cd..
cd ..
echo     Cleaning out user15\mail directory
cd user15
cd mail
copy c:\public\idfile.dat,idfile.dat
if exist temp.fil  ren temp.fil, flag.fil
if exist user15.n* del user15.n*
if exist user15.o* del user15.o*
if exist in.fil    del in.fil
if exist *.mes     del *.mes
cd..
cd ..
echo     Cleaning out user16\mail directory
cd user16
cd mail
copy c:\public\idfile.dat,idfile.dat
if exist temp.fil  ren temp.fil, flag.fil
if exist user16.n* del user16.n*
if exist user16.o* del user16.o*
if exist in.fil    del in.fil
if exist *.mes     del *.mes
cd..
cd ..
```

```
rem  ****** NOTE ******
rem  To extend # of users from 16 to 32 remove all rems from this point on

rem echo     Cleaning out user17\mail directory
rem cd user17
rem cd mail
rem copy c:\public\idfile.dat,idfile.dat
rem if exist temp.fil  ren temp.fil, flag.fil
rem if exist user17.n* del user17.n*
rem if exist user17.o* del user17.o*
rem if exist in.fil    del in.fil
rem if exist *.mes     del *.mes
rem cd..
rem cd ..
rem echo     Cleaning out user18\mail directory
rem cd user18
rem cd mail
rem copy c:\public\idfile.dat,idfile.dat
rem if exist temp.fil  ren temp.fil, flag.fil
rem if exist user18.n* del user18.n*
rem if exist user18.o* del user18.o*
rem if exist in.fil    del in.fil
rem if exist *.mes     del *.mes
rem cd..
rem cd ..
rem echo     Cleaning out user19\mail directory
rem cd user19
rem cd mail
rem copy c:\public\idfile.dat,idfile.dat
rem if exist temp.fil  ren temp.fil, flag.fil
rem if exist user19.n* del user19.n*
rem if exist user19.o* del user19.o*
rem if exist in.fil    del in.fil
rem if exist *.mes     del *.mes
rem cd..
rem cd ..
rem echo     Cleaning out user20\mail directory
rem cd user20
rem cd mail
rem copy c:\public\idfile.dat,idfile.dat
rem if exist temp.fil  ren temp.fil, flag.fil
rem if exist user20.n* del user20.n*
rem if exist user20.o* del user20.o*
rem if exist in.fil    del in.fil
rem if exist *.mes     del *.mes
rem cd..
rem cd ..
rem echo     Cleaning out user21\mail directory
rem cd user21
rem cd mail
rem copy c:\public\idfile.dat,idfile.dat
rem if exist temp.fil  ren temp.fil, flag.fil
rem if exist user21.n* del user21.n*
rem if exist user21.o* del user21.o*
rem if exist in.fil    del in.fil
rem if exist *.mes     del *.mes
rem cd..
rem cd ..
rem echo    Cleaning out user22\mail directory
rem cd user22
rem cd mail
rem copy c:\public\idfile.dat,idfile.dat
rem if exist temp.fil  ren temp.fil, flag.fil
rem if exist user22.n* del user22.n*
rem if exist user22.o* del user22.o*
rem if exist in.fil    del in.fil
rem if exist *.mes     del *.mes
rem cd..
rem cd ..
rem echo    Cleaning out user23\mail directory
rem ced user23
rem cd mail
rem copy c:\public\idfile.dat,idfile.dat
rem if exist temp.fil  ren temp.fil, flag.fil
rem if exist user23.n* del user23.n*
rem if exist user23.o* del user23.o*
rem if exist in.fil    del in.fil
rem if exist *.mes     del *.mes
rem cd..
rem cd ..
rem echo    Cleaning out user24\mail directory
rem cd user24
rem cd mail
rem copy c:\public\idfile.dat,idfile.dat
rem if exist temp.fil  ren temp.fil, flag.fil
rem if exist user24.n* del user24.n*
rem if exist user24.o* del user24.o*
rem if exist in.fil    del in.fil
rem if exist *.mes     del *.mes
rem cd..
rem cd ..
rem echo    Cleaning out user25\mail directory
rem cd user25
rem cd mail
rem copy c:\public\idfile.dat,idfile.dat
rem if exist temp.fil  ren temp.fil, flag.fil
rem if exist user25.n* del user25.n*
rem if exist user25.o* del user25.o*
rem if exist in.fil    del in.fil
rem if exist *.mes     del *.mes
rem cd..
rem cd ..
rem echo    Cleaning out user26\mail directory
rem cd user26
rem cd mail
rem copy c:\public\idfile.dat,idfile.dat
rem if exist temp.fil  ren temp.fil, flag.fil
rem if exist user26.n* del user26.n*
rem if exist user26.o* del user26.o*
rem if exist in.fil    del in.fil
rem if exist *.mes     del *.mes
rem cd..
rem cd ..
rem echo     Cleaning out user27\mail directory
rem cd user27
rem cd mail
rem copy c:\public\idfile.dat,idfile.dat
rem if exist temp.fil  ren temp.fil, flag.fil
rem if exist user27.n* del user27.n*
rem if exist user27.o* del user27.o*
rem if exist in.fil    del in.fil
rem if exist *.mes     del *.mes
rem cd..
rem cd ..
rem echo     Cleaning out user28\mail directory
rem cd user28
rem cd mail
rem copy c:\public\idfile.dat,idfile.dat
rem if exist temp.fil  ren temp.fil, flag.fil
rem if exist user28.n* del user28.n*
rem if exist user28.o* del user28.o*
rem if exist in.fil    del in.fil
rem if exist *.mes     del *.mes
rem cd..
rem cd ..
rem echo     Cleaning out user29\mail directory
rem cd user29
rem cd mail
rem copy c:\public\idfile.dat,idfile.dat
rem if exist temp.fil  ren temp.fil, flag.fil
rem if exist user29.n* del user29.n*
rem if exist user29.o* del user29.o*
rem if exist in.fil    del in.fil
rem if exist *.mes     del *.mes
rem cd..
rem cd ..
rem echo     Cleaning out user30\mail directory
rem cd user30
rem cd mail
rem copy c:\public\idfile.dat,idfile.dat
rem if exist temp.fil  ren temp.fil, flag.fil
rem if exist user30.n* del user30.n*
rem if exist user30.o* del user30.o*
rem if exist in.fil    del in.fil
rem if exist *.mes     del *.mes
rem cd..
rem cd ..
rem echo     Cleaning out user31\mail directory
rem cd user31
rem cd mail
rem copy c:\public\idfile.dat,idfile.dat
rem if exist temp.fil  ren temp.fil, flag.fil
rem if exist user31.n* del user31.n*
rem if exist user31.o* del user31.o*
rem if exist in.fil    del in.fil
rem if exist *.mes     del *.mes
rem cd..
rem cd ..
rem echo     Cleaning out user32\mail directory
rem cd user32
rem cd mail
rem copy c:\public\idfile.dat,idfile.dat
rem if exist temp.fil  ren temp.fil, flag.fil
rem if exist user32.n* del user32.n*
rem if exist user32.o* del user32.o*
rem if exist in.fil    del in.fil
rem if exist *.mes     del *.mes
rem cd..
rem cd ..

if exist email.log del email.log
if log == %1 copy email.dum email.log

cd\sessman
cls
echo *********************************************************
echo *                                                       *
echo * INITIALIZE SYSTEM TIME FILE (takes approx. 5 seconds) *
echo *                                                       *
echo *********************************************************
gtime
echo *********************************************************
echo *                                                       *
echo *        INITIALIZATION OF EMAIL SYSTEM COMPLETE        *
echo *                                                       *
echo *********************************************************
echo on
cycle
```

APPENDIX N

PROGRAM LISTING

of

GOFILE.BAT

```
echo off
rem  **********************************************************************
rem  *                                                                    *
rem  *  GOFILE.BAT                                                        *
rem  *  This bat file is used to copy IDFILE.DAT to each user's MAIL      *
rem  *  subdirectory.  IDFILE.DAT provides the mapping between username   *
rem  *  and location on the network.  It is used by the EMAIL system      *
rem  *  to allow communication between users but relieve users from       *
rem  *  having to know the actual location of the receiver on the net-    *
rem  *  work.  This bat file is called from the EMAIL Individual Tools    *
rem  *  menu located in the SESSMAN directory.  Before copying a file to  *
rem  *  the user's mail subdirectory, a procedure LOCK is called to pre-  *
rem  *  vent the user's EMAIL program from accessing IDFILE.DAT while it  *
rem  *  is being updated/copied.  After the copy is done a procedure      *
rem  *  UNLOCK is executed to allow the user's EMAIL system to access the *
rem  *  updated IDFILE.DAT.                                               *
rem  *                                                                    *
rem  **********************************************************************
cls
cd..
cd public
echo off
echo **********************************************************
echo *                                                        *
echo *       COPY IDFILE.DAT TO USER MAIL DIRECTORIES.        *
echo *                                                        *
echo **********************************************************
echo       Copying IDFILE.DAT to user1\mail directory
cd user1
cd mail
lock
copy c:\public\idfile.dat, idfile.dat
unlock
cd..
cd ..
echo       Copying IDFILE.DAT to user2\mail directory
cd user2
cd mail
lock
copy c:\public\idfile.dat, idfile.dat
unlock
cd..
cd ..
echo       Copying IDFILE.DAT to user3\mail directory
cd user3
cd mail
lock
copy c:\public\idfile.dat, idfile.dat
unlock
cd..
cd ..
echo       Copying IDFILE.DAT to user4\mail directory
cd user4
cd mail
lock
copy c:\public\idfile.dat, idfile.dat
unlock
cd..
cd ..
echo       Copying IDFILE.DAT to user5\mail directory
cd user5
cd mail
lock
copy c:\public\idfile.dat, idfile.dat
unlock
cd..
cd ..
echo       Copying IDFILE.DAT to user6\mail directory
cd user6
cd mail
lock
copy c:\public\idfile.dat, idfile.dat
unlock
cd..
cd ..
echo       Copying IDFILE.DAT to user7\mail directory
cd user7
cd mail
lock
copy c:\public\idfile.dat, idfile.dat
unlock
cd..
cd ..
echo       Copying IDFILE.DAT to user8\mail directory
cd user8
cd mail
lock
copy c:\public\idfile.dat, idfile.dat
unlock
cd..
cd ..
echo       Copying IDFILE.DAT to user9\mail directory
cd user9
cd mail
lock
copy c:\public\idfile.dat, idfile.dat
unlock
cd..
cd ..
echo       Copying IDFILE.DAT to user10\mail directory
cd user10
cd mail
lock
copy c:\public\idfile.dat, idfile.dat
unlock
cd..
cd ..
```

```
echo      Copying IDFILE.DAT to user11\mail directory
cd user11
cd mail
lock
copy c:\public\idfile.dat, idfile.dat
unlock
cd..
cd ..
echo      Copying IDFILE.DAT to user12\mail directory
cd user12
cd mail
lock
copy c:\public\idfile.dat, idfile.dat
unlock
cd..
cd ..
echo      Copying IDFILE.DAT to user13\mail directory
cd user13
cd mail
lock
copy c:\public\idfile.dat, idfile.dat
unlock
cd..
cd ..
echo      Copying IDFILE.DAT to user14\mail directory
cd user14
cd mail
lock
copy c:\public\idfile.dat, idfile.dat
unlock
cd..
cd ..
echo      Copying IDFILE.DAT to user15\mail directory
cd user15
cd mail
lock
copy c:\public\idfile.dat, idfile.dat
unlock
cd..
cd ..
echo      Copying IDFILE.DAT to user16\mail directory
cd user16
cd mail
lock
copy c:\public\idfile.dat, idfile.dat
unlock
cd..
cd ..

rem  ###### NOTE ######
rem  To extend # of users from 16 to 32 remove all rems from this point on

rem echo      Copying IDFILE.DAT to user17\mail directory
rem cd user17
rem cd mail
rem lock
rem copy c:\public\idfile.dat, idfile.dat
rem unlock
rem cd..
rem cd ..
rem echo      Copying IDFILE.DAT to user18\mail directory
rem cd user18
rem cd mail
rem lock
rem copy c:\public\idfile.dat, idfile.dat
rem unlock
rem cd..
rem cd ..
rem echo      Copying IDFILE.DAT to user19\mail directory
rem cd user19
rem cd mail
rem lock
rem copy c:\public\idfile.dat, idfile.dat
rem unlock
rem cd..
rem cd ..
rem echo      Copying IDFILE.DAT to user20\mail directory
rem cd user20
rem cd mail
rem lock
rem copy c:\public\idfile.dat, idfile.dat
rem unlock
rem cd..
rem cd ..
rem echo      Copying IDFILE.DAT to user21\mail directory
rem cd user21
rem cd mail
rem lock
rem copy c:\public\idfile.dat, idfile.dat
rem unlock
rem cd..
rem cd ..
rem echo      Copying IDFILE.DAT to user22\mail directory
rem cd user22
rem cd mail
rem lock
rem copy c:\public\idfile.dat, idfile.dat
rem unlock
rem cd..
rem cd ..
rem echo      Copying IDFILE.DAT to user23\mail directory
rem ced user23
rem cd mail
rem lock
rem copy c:\public\idfile.dat, idfile.dat
rem unlock
rem cd..
rem cd ..
rem echo      Copying IDFILE.DAT to user24\mail directory
rem cd user24
rem cd mail
rem lock
rem copy c:\public\idfile.dat, idfile.dat
rem unlock
rem cd..
rem cd ..
rem echo      Copying IDFILE.DAT to user25\mail directory
rem cd user25
rem cd mail
rem lock
rem copy c:\public\idfile.dat, idfile.dat
rem unlock
rem cd..
rem cd ..
rem echo      Copying IDFILE.DAT to user26\mail directory
rem cd user26
rem cd mail
rem lock
rem copy c:\public\idfile.dat, idfile.dat
rem unlock
rem cd..
rem cd ..
rem echo      Copying IDFILE.DAT to user27\mail directory
rem cd user27
rem cd mail
rem lock
rem copy c:\public\idfile.dat, idfile.dat
rem unlock
rem cd..
rem cd ..
rem echo      Copying IDFILE.DAT to user28\mail directory
rem cd user28
rem cd mail
rem lock
rem copy c:\public\idfile.dat, idfile.dat
rem unlock
rem cd..
rem cd ..
rem echo      Copying IDFILE.DAT to user29\mail directory
rem cd user29
rem cd mail
rem lock
rem copy c:\public\idfile.dat, idfile.dat
rem unlock
rem cd..
rem cd ..
rem echo      Copying IDFILE.DAT to user30\mail directory
rem cd user30
rem cd mail
rem lock
rem copy c:\public\idfile.dat, idfile.dat
rem unlock
rem cd..
rem cd ..
rem echo       Copying IDFILE.DAT to user31\mail directory
rem cd user31
rem cd mail
rem lock
rem copy c:\public\idfile.dat, idfile.dat
rem unlock
rem cd..
rem cd ..
rem echo       Copying IDFILE.DAT to user32\mail directory
rem cd user32
rem cd mail
rem lock
rem copy c:\public\idfile.dat, idfile.dat
rem unlock
rem cd..
rem cd ..

echo *********************************************************
echo *                                                       *
echo * PLACEMENT OF IDFILE.DAT IN USER DIRECTORIES COMPLETE  *
echo *                                                       *
echo *********************************************************
echo on
cd\sessman
cycle
```

APPENDIX O

PROGRAM LISTING

of

INCLUDE FILES USED

by the

MEMORY-RESIDENT SHELL:

StayRstr.420
StaySave.420
StayI28.410
StayI8.420
StayI21.410
StayI13.410
StayI16.410
StaySubs.420
StayWndo.341
ClkI8.410

```
Inline(
                              {;$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$;}
                              {;                  P R O C E S S   I N T E R R U P T  1 6              ;}
                              {;                                                                      ;}
                              {;$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$;}
                              {; Function:}
                              {;          Provide a Keyboard trap to allow concurrent processes to}
                              {;          run in the background while a Turbo Read is active.}
                              {;}
                              {; Copyright (C) 1985,1986}
                              {;                    Lane Ferris}
                              {;              - The Hunter's Helper -}
                              {;       Distributed to the Public Domain for use without profit.}
                              {;              Original Version 5.15.85}
                              {;}
                              {;          ; On entry the Stack will already contain: ;}
                              {;          ;  1) Sp for Dos                           ;}
                              {;          ;  2) Bp for Dos                           ;}
                              {;          ;  3) Ip for Dos                           ;}
                              {;          ;  4) Cs for Dos                           ;}
                              {;          ;  5) Flags for Dos                        ;}
  $5D                         {        Pop     Bp}
  /$5D                        {        Pop     Bp               ; Restore Original Bp}
  /$80/$FC/$00                {        Cmp     Ah,00            ; If Char request,}
  /$74/$2A                    {        Je      Func00           ;  loop for character}
  /$80/$FC/$01                {        Cmp     Ah,01            ; If character availability test}
  /$74/$05                    {        Je      Func01           ;   go check for char}
                              {GoBios16:}
  /$2E                        {        CS:                      ; Go to Bios Interrupt 16}
  /$FF/$2E/>BIOS_INT16        {        Jmp Far [>BIOS_Int16]}
                              {Func01:}
  /$E8/$3F/$00                {        Call    KeyStat          ; Look at Key buffer}
  /$9C                        {        PushF}
  /$74/$16                    {        Jz      Fret01           ; Return if no key}
  /$2E                        {        CS:                      ; Test for HOT KEY}
  /$3A/$26/>OUR_HOTKEY        {        Cmp     Ah,[<Our_HotKey]}
  /$75/$0F                    {        Jne     Fret01}
  /$B4/$00                    {        Mov     Ah,0             ; Remove the HotKey}
  /$2E                        {        CS:                      ;  flags are removed by BIOS return}
  /$FF/$1E/>BIOS_INT16        {        Call Dword [>BIOS_INT16]}
  /$2E                        {        CS:                      ; Say we saw the HOT Key}
  /$80/$0E/>STATUS/<HOTKEY_ON {        Or by [<Status],<HotKey_ON}
  /$EB/$E4                    {        Jmp     Func01           ;}
                              {Fret01:}
  /$9D                        {        POPF}
  /$CA/$02/$00                {        RETF    2                ; Return to user}
                              {Func00:}
  /$E8/$1F/$00                {        Call    KeyStat          ; Wait until character available}
  /$74/$FB                    {        Jz      Func00}
  /$B4/$00                    {        Mov     Ah,0             ; Get the next User Key}
  /$9C                        {        PUSHF                    ;}
  /$2E                        {        CS:}
  /$FF/$1E/>BIOS_INT16        {        Call Dword [>BIOS_INT16]}
  /$9C                         {       PushF                   ; Save Return Flags}
  /$2E                         {       CS:}
  /$3A/$26/>OUR_HOTKEY         {       Cmp     Ah,[<Our_HotKey]; Our HotKey ?}
  /$74/$04                     {       Je      GotHotKey       ; yes..enter Staysave code}
  /$9D                         {       POPF                    ; else Restore INT 16 flags}
  /$CA/$02/$00                 {       RetF    2               ; Return W/Key discard original INT 16 flags}
                               {                               ; ".. give it to Mikey..he'll eat anything"}
                               {GotHotKey:}
  /$9D                         {       POPF                    ; Discard INT16 return flags}
  /$2E                         {       CS:                     ; Say we saw the HOT Key}
  /$80/$0E/>STATUS/<HOTKEY_ON  {       Or by [<Status],<HotKey_ON}
  /$EB/$DE                     {       Jmp  Func00             ; Get another Key}
                               {;}
                               {;      Call the Background task if no key is available}
                               {;}
                               {KeyStat:}
  /$B4/$01                     {       Mov    Ah,01            ; Look for available key}
  /$9C                         {       Pushf                   ; Call the keyboard function}
  /$2E                         {       CS:}
  /$FF/$1E/>BIOS_INT16         {       Call dw [<BIOS_INT16]}
  /$74/$01                     {       Jz      ChkDosCr       ; No Character available from Keyboard}
  /$C3                         {       RET                    ; else return with new flags and code}
                               {ChkDosCr:}
  /$06                         {       Push    ES             ; Check if DOS Critical error in effect}
  /$56                         {       Push    Si}
  /$2E                         {       CS:}
  /$C4/$36/>DOSSTAT2           {       Les     Si,[>DOSStat2]}
  /$26                         {       ES:                     ; Zero says DOS is interruptable}
  /$AC                         {       Lodsb                   ; $FF  says Dos is in a critical state}
  /$2E                         {       CS:}
  /$C4/$36/>DOSSTAT1           {       Les     Si,[>DosStat1]  ; If INDOS then INT $28 issued by DOS}
  /$26                         {       ES:                     ; so we dont have to.}
  /$0A/$04                     {       Or      Al,[SI]}
  /$2E                         {       CS:                     ; Account for active interrupts}
  /$0A/$06/>INTR_FLAGS         {       Or      Al,[<Intr_Flags]; Any flags says we dont issue call}
  /$5E                         {       Pop     Si              ; to the background.}
  /$07                         {       Pop     Es}
  /$3C/$01                     {       Cmp     Al,01           ; Must be INDOS flag only}
  /$7F/$02                     {       JG      Skip28          ; DOS cannot take an interrupt yet}
  /$CD/$28                     {       INT     $28             ; Call Dos Idle func. (background dispatch).}
                               {Skip28:}
  /$31/$C0                     {       Xor     Ax,Ax           ; Show  no keycode available}
  /$C3                         {       RET}
                               {;----------------------------------------------------------------}
);
```

```
Inline(
                              {; STAYI21.400}
                              {;-----------}
                              {; Routine to Set a Flag when certain INT21 functions are active.}
                              {; Functions to be flagged are identified in the main Stayres}
                              {; routine. Cf. Functab array.}
 $5D                          {         Pop   Bp                      ; Remove Turbo Prologue}
 /$5D                         {         Pop   Bp}
 /$9C                         {         PushF}
 /$FB                         {         STI                           ; Allow interrupts}
 /$80/$FC/$62                 {         Cmp   Ah,$62                  ; Verify Max function}
 /$7F/$28                     {         Jg    SkipI21}
                              {; Some Int 21 functions must be left alone. They either never return,}
                              {; grab parameters from the stack, or can be interrupted. This code}
                              {; takes account of those possibilities.}
 /$50                         {         Push  Ax                      ; Skip functions marked 1 in}
 /$53                         {         Push  Bx                      ; in the function table.}
 /$86/$C4                     {         Xchg  Ah,Al}
 /$BB/>FUNCTAB                {         Mov   Bx,>FuncTab             ; Test Int 21 function}
 /$2E                         {         CS:}
 /$D7                         {         Xlat}
 /$08/$C0                     {         Or    Al,Al                   ; Wait for functions marked zero}
 /$5B                         {         Pop   Bx                      ; in the function table.}
 /$58                         {         Pop   Ax}
 /$75/$19                     {         Jnz   SkipI21}
                              {SetI21:}
 /$2E                         {         CS:}
 /$80/$0E/>INTR_FLAGS/<INT21_ON{          Or by [<Intr_flags],<INT21_on ; Say INT 21 is Active}
 /$9D                         {         PopF}
 /$9C                         {         Pushf}
 /$2E                         {         CS:}
 /$FF/$1E/>DOS_INT21          {         Call dw [<DOS_INT21]          ; Invoke Original INT 21}
 /$FB                         {         STI                           ; Insure interrupts enabled}
 /$9C                         {         Pushf                         ; Save Return Flags}
 /$2E                         {         CS:                           ; Clear INT 21 Active}
 /$80/$26/>INTR_FLAGS/<FOXS-INT21_ON{        And by [<Intr_flags],<Foxs-INT21_on}
 /$9D                         {         Popf                          ; Retrieve the flags}
 /$CA/$02/$00                 {         RETF  2}
                              {SkipI21:                               ; Invoke Int 21 w/o return}
 /$9D                         {         PopF}
 /$2E                         {         CS:}
 /$FF/$2E/>DOS_INT21          {         Jmp dw [>Dos_INT21]}
                              {;...................................................................}
);
```

```
Inline(
                              {; STAYI28.400}
                              {;------------}
                              {; Routine to Invoke User Code When HotKey or DOS idle}
  $5D                         {         Pop   Bp                      ; Remove Turbo Prologue}
  /$5D                        {         Pop   Bp}
  /$9C                        {         Pushf}
  /$2E                        {         CS:}
  /$FF/$1E/>DOS_INT28         {         Call dw [>DOS_INT28]          ; Invoke Original INT 28}
  /$2E                        {         CS:}
  /$F6/$06/>STATUS/<HOTKEY_ON {         Test by [<Status],<HotKey_on  ; Have we received the HOKEY}
  /$74/$25                    {         Jz    NoGo}
  /$2E                        {         CS:}
  /$F6/$06/>STATUS/<INUSE     {         Test by [<Status],<Inuse      ; If Inuse.. then No go}
  /$75/$1D                    {         Jnz   NoGo}
                              {; If Not already waiting I/O, not already in use, and HotKey received}
                              {; see if DOS is now interruptable}
                              {ChkIO:}
  /$06                        {         Push  ES                      ; Save registers}
  /$56                        {         Push  Si}
  /$50                        {         Push  Ax}
  /$2E                        {         CS:}
  /$C4/$36/>DOSSTAT2          {         LES   SI,[>DOSstat2]          ; Fetch DOS Critical status byte}
  /$26                        {         ES:}
  /$AC                        {         LodSb}
  /$2E                        {         CS:}
  /$0A/$06/>INTR_FLAGS        {         Or    Al,[<Intr_Flags]        ; Add Interrupt active flags}
  /$58                        {         Pop   Ax}
  /$5E                        {         Pop   Si}
  /$07                        {         Pop   ES}
  /$75/$09                    {         Jnz   NoGo                      ; Wait for inactivity}
  /$2E                        {         CS:                           ; Have the HotKey}
  /$80/$3E/>WAITCOUNT/$00     {         Cmp by [<WaitCount],00        ; If timer waiting, go}
  /$E9/$01/$00                {         Jmp Go}
                              {NoGo:}
  /$CF                        {         IRET}
                              {GO:                                    ; Enter the User's Turbo Procedure}
  /$2E                        {         CS:}
  /$C6/$06/>WAITCOUNT/$00     {         Mov by [<WaitCount],00        ; Kill INT8 wait count}
  /$2E                        {         CS:}
  /$FF/$16/>USERPROGRAM       {         Call [<UserProgram]}
  /$CF                        {         IRET}
                              {;......................................................................}
);
```

```
Inline(
                            {; STAYI8.413}
                            {;-----------}
                            {; Routine to Await Outstanding I/O, then post Stayres Active}
  $5D                       {    Pop   Bp                    ; Remove Turbo Prologue}
  /$5D                      {    Pop   Bp}
  /$9C                      {    Pushf}
  /$2E                      {    CS:}
  /$FF/$1E/>BIOS_INT8       {    Call dw [>BIOS_INT8]        ; Invoke Original INT 8}
  /$2E                      {    CS:}
  /$F6/$06/>STATUS/<HOTKEY_ON {  Test by [<Status],<HotKey_on  ; Have we received the HOKEY}
  /$74/$39                  {    Jz    NoGo}
  /$2E                      {    CS:}
  /$F6/$06/>STATUS/<INUSE   {    Test by [<Status],<Inuse    ; If Inuse.. then No go}
  /$75/$31                  {    Jnz   NoGo}
  /$2E                      {    CS:                         ; Have the HotKey}
  /$80/$3E/>WAITCOUNT/$00   {    Cmp by [<WaitCount],00      ; If waiting, check time}
  /$75/$22                  {    Jnz   Waiting}
                            {; If Not already waiting I/O, not already in use, and HotKey received}
                            {; see if DOS is now interruptable}
                            {ChkIO:}
  /$06                      {    Push  ES                    ; Save registers}
  /$56                      {    Push  Si}
  /$50                      {    Push  Ax}
  /$2E                      {    CS:}
  /$C4/$36/>DOSSTAT1        {    LES   Si,[>DOSstat1]        ; Fetch Dos status 1}
  /$26                      {    ES:}
  /$AC                      {    Lodsb                       ; Fetch Status byte from dos}
  /$2E                      {    CS:}
  /$C4/$36/>DOSSTAT2        {    LES   SI,[>DOSstat2]        ; Add second status byte}
  /$26                      {    ES:}
  /$0A/$04                  {    Or    Al,[SI]}
  /$2E                      {    CS:}
  /$0A/$06/>INTR_FLAGS      {    Or    Al,[<Intr_Flags]      ; Add Interrupt active flags}
  /$58                      {    Pop   Ax}
  /$5E                      {    Pop   Si}
  /$07                      {    Pop   ES}
  /$74/$0E                  {    Jz    Go                    ; Wait for inactivity}
  /$2E                      {    CS:}
  /$C6/$06/>WAITCOUNT/$10   {    Mov by [<WaitCount],$10     ; Set Wait count}
                            {Waiting:}
  /$2E                      {    CS:}
  /$FE/$0E/>WAITCOUNT       {    Dec by [<WaitCount]         ; Decrement wait count}
  /$74/$D7                  {    Jz   ChkIO}
                            {NoGo:}
  /$CF                      {    IRET}
                            {GO:   ; Enter the User's Turbo Procedure}
  /$2E                      {    CS:}
  /$FF/$16/>USERPROGRAM     {    Call [<UserProgram]}
  /$CF                      {    IRET}
                            {;.................................................................}
);
```

```
Inline(

{;$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$;}
{;                  S T A Y R S T R . I N C                                   ;}
{;                                                                            ;}
{;           This is the StayRstr.Inc file included above                     ;}
{;$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$;}
                              {;Version 4.15}
                              {      ; Inline Code to restore the stack and regs moved}
                              {      ; to the Turbo Resident Stack which allows}
                              {      ; Turbo Terminate & Stay Resident programs.}
                              {      ; Copr. 1985, 1986}
                              {      ; Author: Lane Ferris}
                              {      ;        - The Hunter's Helper -}
                              {      ; Distributed to the Public Domain for use without profit.}
                              {      ; Original Version 5.15.85}
                              {;---------------------------------------------------------------------;}
                              {;        Restore the Dos (or interrupted pgm) Regs and Stack          ;}
                              {;---------------------------------------------------------------------;}
                              {; Replace the Users Saved Stack}
                              {; Note that pushes on the stack go in the opposite direction of our}
                              {; moves. Thus we dont worry about REP stack activity overlaying the}
                              {; enabled REP fuction.}
  $FA                         {    CLI}
  /$2E                        {    CS:     ;Avoid stack manipulation if never "StaySaved"}
  /$A1/>DOSSSIZ               {    Mov  Ax,[>DosSsiz]}
  /$09/$C0                    {    Or   Ax,Ax}
  /$74/$20                    {    Jz   NotinDos}
  /$8C/$D0                    {    Mov  Ax,SS         ;Source is our Stack}
  /$8E/$D8                    {    Mov  DS,Ax}
  /$89/$E6                    {    Mov  Si,Sp         ;Point to Last used USER word on our stack}
  /$46                        {    Inc  Si}
  /$46                        {    Inc  Si}
  /$2E                        {    CS:}
  /$8E/$06/>DOSSSEG           {    Mov  ES,[>DosSSeg] ;Dest is Dos indos primary Stack}
  /$2E                        {    CS:}
  /$8B/$3E/>DOSSPTR           {    Mov  Di,[>DosSptr]}
  /$2E                        {    CS:}
  /$8B/$0E/>DOSSSIZ           {    Mov  Cx,[>DosSsiz] ;Saved words}
  /$29/$CF                    {    Sub  Di,Cx         ;point to last used word of Dos stack}
  /$29/$CF                    {    Sub  Di,Cx}
  /$FC                        {    CLD}
  /$F2/$A5                    {    Rep Movsw          ;Careful! Interrupt are enabled here}
  /$89/$F4                    {    Mov  Sp,Si         ;Skip over moved words}
                              {;--}
                              {NotinDos:}
  /$07                        {    Pop  Es}
  /$5F                        {    Pop  Di}
  /$5E                        {    Pop  Si}
  /$5A                        {    Pop  Dx}
  /$59                        {    Pop  Cx}
  /$5B                        {    Pop  Bx}
  /$58                        {    Pop  Ax}
  /$2E                        {    CS:}
  /$80/$26/>STATUS/<FOXS-INUSE-HOTKEY_ON{   And by [<Status],<Foxs-Inuse-HotKey_on   ; Clear INUSE flag}
  /$2E                        {    CS:                                     ; .. and HotKey}
  /$BE/$1E/>USRDSEG           {    Mov  DS,[<UsrDSeg]}
  /$2E                        {    CS:}
  /$BE/$16/>USRSSEG           {    Mov  SS,[<UsrSSeg]}
  /$2E                        {    CS:}
  /$8B/$26/>USRSPTR           {    Mov  SP,[<UsrSPtr]}
  /$5D                        {    Pop  Bp             ; Remove Bp,Sp from Procedure entry}
  /$5D                        {    Pop  Bp}
  /$FB                        {    STI                 ; enable interrupts}
  /$C3                        {    RET}
                              {;.......................................................................}
);
```

```
Inline(

{;######################################################################;}
{;                 S T A Y S A V E . I N C                             ;}
{;######################################################################;}
                      {;Version 4.15}
                      {;}
                      {; This Inline routine will save the regs and Stack for Stay resident programs.}
                      {; It restores DS and SS from the previously saved integer constants "OurDseg"}
                      {; and "OurSSeg".  DS is restored from the Turbo Initialization Savearea.}
                      {; Author: Copyr. 1985, 1986}
                      {;               Lane Ferris}
                      {;             - The Hunter's Helper -}
                      {;       Distributed to the Public Domain for use without profit.}
                      {;                   Original Version 5.15.85}
 $FA                  {       CLI                   ; Stop all interrupts}
 /$2E                 {       CS:}

 /$80/$0E/>STATUS/<INUSE   {       Or by   [<Status],<InUse  ; Set Active bit}
                           {; Switch the SS:Sp reg pair over to ES:Si}
                           {; Put Turbo's Stack pointers into SS:Sp}
 /$2E                      {       CS:}
 /$8C/$1E/>USRDSEG         {       Mov     [>UsrDSeg],DS       ; Save Usr DataSegment}
 /$2E                      {       CS:}
 /$8C/$16/>USRSSEG         {       Mov     [>UsrSSeg],SS       ; Save Usr Stack Segment}
 /$2E                      {       CS:}
 /$89/$26/>USRSPTR         {       Mov     [>UsrSPtr],Sp       ; Save Usr Stack Ptr}
                           {; Stack User interrupted pgm regs for Exit.}
                           {; These are the original interrupt process regs}
                           {; that must be returned on interrupt return}
 /$2E                      {       CS:}
 /$8E/$1E/>OURDSEG         {       Mov     DS,[>OurDseg] ; Get Turbo Stack pointer from DataSegment}
 /$2E                      {       CS:}
 /$8E/$16/>OURSSEG         {       Mov     SS,[>OurSSeg]}
 /$8B/$26/$74/$01          {       Mov     Sp,[$174]    ; Sp set by code at $B2B in Turbo initialization}
 /$55                      {       Push    Bp}
 /$50                      {       Push    Ax}
 /$53                      {       Push    Bx}
 /$51                      {       Push    Cx}
 /$52                      {       Push    Dx}
 /$56                      {       Push    Si}
 /$57                      {       Push    Di}
 /$06                      {       Push    Es}
                           {; Save the InDOS stack to avoid recursion crashes (Writeln).}
                           {; Setup destination to Turbo Stack}
 /$89/$E7                  {       Mov     Di,Sp  ;Dest is our stack}
 /$4F                      {       Dec     Di     ;Back off current used word}
 /$4F                      {       Dec     Di}
 /$2E                      {       CS:}
 /$8C/$D0                  {       Mov     Ax,SS  ;Turbo stack is destination}
 /$8E/$C0                  {       Mov     ES,Ax}
                           {; Setup source from DOS Indos primary stack}
 /$2E                      {       CS:}
```

```
    /$8E/$1E/>DOSSSEG          {       Mov     DS,[>DosSSeg]   ; Source is DOS Indos primary stack}
    /$2E                       {       CS:}
    /$8B/$36/>DOSSPTR          {       Mov     Si,[>DosSptr]   ; DOS primary stack offset}
    /$B9/$40/$00               {       Mov     Cx,$40}
    /$2E                       {       CS:}
    /$89/$0E/>DOSSSIZ          {       Mov     [>DosSsiz],Cx ;remember the stack word size}
    /$4E                       {       Dec     Si      ;point last word on stack}
    /$4E                       {       Dec     Si}
    /$89/$E0                   {       Mov     Ax,Sp           ;Get stack pointer higher to avoid}
    /$29/$C8                   {       Sub     Ax,Cx           ;overwriting during enabled REP functions}
    /$29/$C8                   {       Sub     Ax,Cx}
    /$89/$C4                   {       Mov     Sp,Ax}
    /$FD                       {       STD             ;Move like Pushes on stack}
    /$F2/$A5                   {       Rep Movsw       ;Move users stack to our own}
    /$89/$FC                   {       Mov     Sp,Di   ;Update our stack pointer to available word.}
    /$FC                       {       Cld}
    /$2E                       {       CS:}
    /$8E/$1E/>OURDSEG          {       Mov     DS,[>OurDSeg]   ; Setup Turbo Data Segment Pointer}
    /$FB                       {       STI                     ; Enable Interrupts}
                               {;......................................................................}
  );
```

```
{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
{                         S T A Y S U B S . I N C                            }
{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
      {--------------------------------------------------------}
      {            S E T U P   I N T E R R U P T               }
      {--------------------------------------------------------}{
{        Msg # $48   Dated 07-07-86 16:54:36
         From: NEIL RUBENKING
         To: LANE FERRIS
         Re: STAY, WON'T YOU?

         Lane,
              Here's what I did: }

  PROCEDURE Setup_Interrupt(IntNo :byte; VAR IntVec :vector; offset :integer);
  BEGIN
    Regs.Ax := $3500 + IntNo;
    Intr(DosI21,Regs);            {get the address of interrupt }
    IntVec.IP := Regs.BX;         { Location of Interrupt Ip }
    IntVec.CS := Regs.Es;         { Location of Interrupt Cs }
    Regs.Ax := $2500 + IntNo;     { set the interrupt to point to}
    Regs.Ds := Cseg;              {  our procedure}
    Regs.Dx := Offset;
    Intr (DosI21,Regs);
  END;
{$$$$$$$$$$$$$$$$$$$$ C O M M E N T $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
{in the main part of the program}
      Setup_Interrupt(BIOSI16, BIOS_Int16, Ofs(Stay_INT16)); {keyboard}
      Setup_Interrupt(BIOSI10, BIOS_Int10, Ofs(Stay_INT10)); {video}
      Setup_Interrupt(BIOSI8, BIOS_Int8, Ofs(Stay_INT8));    {timer}
      Setup_Interrupt(BIOSI13, BIOS_Int13, Ofs(Stay_INT13)); {disk}
      Setup_Interrupt(DOSI21, DOS_Int21, Ofs(Stay_INT21));   {DOSfunction}
      Setup_Interrupt(DOSI28, DOS_Int28, Ofs(Stay_INT28));   {DOS idle}
$$$$$$$$$$$$$$$$$$$$$$ C O M M E N T $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
      {--------------------------------------------------------}
      {                 S E T   D T A                          }
      {--------------------------------------------------------}
   Procedure SetDTA(var segment, offset : integer );
   BEGIN
     regs.ax := $1A00;       { Function used to get current DTA address }
     regs.Ds := segment;     { Segment of DTA returned by DOS }
     regs.Dx := offset;      { Offset of DTA returned }
     MSDos( regs );          { Execute MSDos function request }
   END;
      {--------------------------------------------------------}
      {                 G E T   D T A                          }
      {--------------------------------------------------------}
   Procedure GetDTA(var segment, offset : integer );
   BEGIN
     regs.ax := $2F00;       { Function used to get current DTA address }
     MSDos( regs );          { Execute MSDos function request }
     segment := regs.ES;     { Segment of DTA returned by DOS }
     offset  := regs.Bx;     { Offset of DTA returned }
   END;
   {-----------------------------------------------------}
   {              S E T   P S P                          }
   {-----------------------------------------------------}
Procedure SetPSP(var segment : integer );
BEGIN

    { A bug in DOS 2.0, 2.1, causes DOS to clobber its standard stack   }
    { when the PSP get/set functions are issued at the DOS prompt. The  }
    { following checks are made, forcing DOS to use the "critical"      }
    { stack when the TSR enters at the INDOS level.                     }

                                   {If Version less then 3.0 and INDOS set }
If DosVersion < 3 then             { then set the Dos Critical Flag        }
   If Mem[DosStat1.CS:DosStat1.IP] <> 0 then
       Mem[DosStat2.CS:DosStat2.IP] := $FF;

  regs.ax := $5000;       { Function to set current PSP address }
  regs.bx := segment;     { Segment of PSP to be used by DOS }
  MSDos( regs );          { Execute MSDos function request }

                                   {If Version less then 3.0 and INDOS set }
  If DosVersion < 3 then           { then clear the Dos Critical Flag     }
     If Mem[DosStat1.CS:DosStat1.IP] <> 0 then
        Mem[DosStat2.CS:DosStat2.IP] := $00;

END;
   {-----------------------------------------------------}
   {              G E T   P S P                          }
   {-----------------------------------------------------}
Procedure GetPSP(var segment : integer );
BEGIN

    { A bug in DOS 2.0, 2.1, causes DOS to clobber its standard stack   }
    { when the PSP get/set functions are issued at the DOS prompt. The  }
    { following checks are made, forcing DOS to use the "critical"      }
    { stack when the TSR enters at the INDOS level.                     }

                             {If Version less then 3.0 and INDOS set }
If DosVersion < 3 then       { then set the Dos Critical Flag        }
   If Mem[DosStat1.CS:DosStat1.IP] <> 0 then
       Mem[DosStat2.CS:DosStat2.IP] := $FF;

  regs.ax := $5100;       { Function to get current PSP address }
  MSDos( regs );          { Execute MSDos function request }
  segment := regs.Bx;     { Segment of PSP returned by DOS }

                             {IF DOS Version less then 3.0 and INDOS set }
If DosVersion < 3 then       { then clear the Dos Critical Flag     }
   If Mem[DosStat1.CS:DosStat1.IP] <> 0 then
        Mem[DosStat2.CS:DosStat2.IP] := $00;
END;
 {----------------------------------------------------------}
 {      G e t   C o n t r o l   C (break)   V e c t o r     }
 {----------------------------------------------------------}
```

```
Type
    Arrayparam = array [1..2] of integer;
Const
     SavedCtlC: arrayparam = (0,0);
     NewCtlC  : arrayparam = (0,0);
 Procedure GetCtlC(Var SavedCtlC:arrayparam);
    Begin                      {Record the Current Ctrl-C Vector}
       With Regs Do
       Begin
       AX:=$3523;
       MsDos(Regs);
       SavedCtlC[1]:=BX;
       SavedCtlC[2]:=ES;
       End;
    End;
    {---------------------------------------------------------------}
    {       S e t   C o n t r o l   C   V e c t o r                 }
    {---------------------------------------------------------------}
    Procedure IRET;           {Dummy Ctrl-C routine}
       Begin
       inline($5D/$5D/$CF);  {Pop Bp/Pop Bp/Iret}
       end;
 Procedure SetCtlC(Var CtlCptr:arrayparam);
    Begin                      {Set the New Ctrl-C Vector}
       With Regs Do
       Begin
        AX:=$2523;
        DS:=CtlCptr[2];
        DX:=CtlCptr[1];
        MsDos(Regs);
       End;
    End;
{----------------------------------------------------------------------}
{          K e y i n   :   R e a d   K e a b o a r d                   }
{----------------------------------------------------------------------}
Function Keyin: char;          { Get a key from the Keyboard            }
   Var Ch : char;              { If extended key, fold above 127        }
   Begin                       {----------------------------------------}
      Repeat until Keypressed;
      Read(Kbd,Ch);
      if (Ch = Esc) and KeyPressed then
         Begin
         Read(Kbd,Ch);
         Ch := Char(Ord(Ch) + 127);
         End;
      Keyin := Ch;
   End;  {Keyin}
{----------------------------------------------------------------------}
{         B e e p   :   S o u n d   t h e   H o r n                    }
{----------------------------------------------------------------------}
Procedure Beep(N :integer); {------------------------------------------}
   Begin                     {  This routine sounds a tone of frequency }
      Sound(n);              {  N for approximately 100 ms              }
      Delay(100);            {------------------------------------------}
      Sound(n div 2);
      Delay(100);
      Nosound;
      End {Beep} ;

      {--------------------------------------------------------------}
      {                I N T E R R U P T   2 4                       }
      {--------------------------------------------------------------}
{ Version 2.0, 1/28/86
  - Bela Lubkin
     CompuServe 76703,3015

     Apologetically mangled by Lane Ferris

  For MS-DOS version 2.0 or greater, Turbo Pascal 1.0 or greater.

  Thanks to Marshall Brain for the original idea for these routines.
  Thanks to John Cooper for pointing out a small flaw in the code.

  These routines provide a method for Turbo Pascal programs to trap
  MS-DOS interrupt 24 (hex).  INT 24h is called by DOS when a 'critical
  error' occurs, and it normally prints the familiar "Abort, Retry,
  Ignore?" message.

  With the INT 24h handler installed, errors of this type will be passed
  on to Turbo Pascal as an error.  If I/O checking is on, this will cause
  a program crash.  If I/O checking is off, IOResult will return an error
  code.  The global variable INT24Err will be true if an INT 24h error
  has occurred.  The variable INT24ErrorCode will contain the INT 24h
  error code as given by DOS. These errors can be found in the DOS
  Technical Reference Manual.

  It is intended that INT24Result be used in place of IOResult. Calling
  INT24Result clears IOResult.  The simple way to use INT24Result is just
  to check that it returns zerc, and if not, handle all errors the same.
  The more complicated way is to interpret the code.  The integer
  returned by INT24Result can be looked at as two bytes.  By assigning
  INT24Result to a variable, you can then examine the two bytes:
  (Hi(<variable>)-1) will give the DOS critical error code, or
  (<variable> And $FF00) will return an integer from the table listed in
  the INT24Result procedure (two ways of looking at the critical error);
  Lo(<variable>) will give Turbo's IOResult.  A critical error will
  always be reflected in INT24Result, but the IOResult part of
  INT24Result will not necessarily be nonzero; in particular,
  unsuccessful writes to character devices will not register as a Turbo
  I/O error.

  INT24Result should be called after any operation which might cause a
  critical error, if Turbo's I/O checking is disabled.  If it is enabled,
  the program will be aborted except in the above noted case of writes to
  character devices.

  Also note that different versions of DOS and the BIOS seem to react to
  printer errors at vastly different rates.  Be prepared to wait a while
  for anything to happen (in an error situation) on some machines.

  These routines are known to work correctly with: Turbo Pascal 1.00B PC-DOS;
                                                   Turbo Pascal 2.00B PC-DOS;
                                                   Turbo Pascal 2.00B MS-DOS;
                                                   Turbo Pascal 3.01A PC-DOS.
  Other MS-DOS and PC-DOS versions should work.

  Note that Turbo 2.0's normal IOResult codes for MS-DOS DO NOT
  correspond to the I/O error numbers given in Appendix I of the Turbo
  2.0 manual, or to the error codes given in the I/O error nn,
  PC=aaaa/Program aborted message.  Turbo 3.0 IOResult codes do match the
  manual.  Here is a table of the correspondence (all numbers in
  hexadecimal):

  Turbo 2.0 IOResult    Turbo error, Turbo 3.0 IOResult
  ------------------    -----------------------------------------------
     00                   00  none
     01                   90  record length mismatch
     02                   01  file does not exist
     03                   F1  directory is full
     04                   FF  file disappeared
     05                   02  file not open for input
     06                   03  file not open for output
     07                   99  unexpected end of file
     08                   F0  disk write error
     09                   10  error in numeric format
     0A                   99  unexpected end of file
     0B                   F2  file size overflow
     0C                   99  unexpected end of file
     0D                   F0  disk write error
     0E                   91  seek beyond end of file
     0F                   04  file not open
     10                   20  operation not allowed on a logical device
     11                   21  not allowed in direct mode
     12                   22  assign to standard files is not allowed
     --                   F3  Too many open files

  - Bela Lubkin
    CompuServe 76703,3015
    1/28/86 }

Const
  INT24Err: Boolean=False;
  INT24ErrCode: Byte=0;
  OldINT24: Array [1..2] Of Integer=(0,0);

Var
  RegisterSet: Record Case Integer Of
                 1: (AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags: Integer);
                 2: (AL,AH,BL,BH,CL,CH,DL,DH: Byte);
               End;

Procedure INT24;     { Interrupt 24 Service Routine }
```

```
  Begin

    Inline( $2E/$C6/$06/ Int24Err / $01/$50/$89/$F8/$2E/$A2/ Int24ErrCode
            /$58/$B0/$00/$89/$EC/$5D/$CF);

{   Turbo:  PUSH BP                      Save caller's stack frame
            MOV  BP,SP                    Set up this procedure's stack frame
            PUSH BP                       ?
    Inline:
            MOV  BYTE CS:[INT24Err],1     Set INT24Err to True
            PUSH AX
            MOV  AX,DI                    Get INT 25h error code
            MOV  CS:[INT24ErrCode],AL     Save it in INT24ErrCode
            POP  AX
            MOV  AL,0                     Tell DOS to ignore the error
            MOV  SP,BP                    Unwind stack frame
            POP  BP
            IRET                          Let DOS handle it from here }
  End;

       {-----------------------------------------------------------}
       {       I N T  2 4   O N                                    }
       {-----------------------------------------------------------}
          { Grab the Critical error ptr from the previous user}
Procedure INT24On;  { Enable INT 24h trapping }
  Begin
    INT24Err:=False;
    With RegisterSet Do
     Begin
      AX:=$3524;
      MsDos(RegisterSet);

      If (OldINT24[1] Or OldINT24[2])=0 Then
       Begin
        OldINT24[1]:=ES;
        OldINT24[2]:=BX;
       End;
      DS:=CSeg;
      DX:=Ofs(INT24);
      AX:=$2524;
      MsDos(RegisterSet);
     End;
  End;
       {-----------------------------------------------------------}
       {                I N T  2 4   O F F                         }
       {-----------------------------------------------------------}
         { Give Critical Error Service pointer back to previous user }
Procedure INT24Off;
  Begin
    INT24Err:=False;
    If OldINT24[1]<>0 Then
      With RegisterSet Do
       Begin
        DS:=OldINT24[1];
        DX:=OldINT24[2];
        AX:=$2524;
        MsDos(RegisterSet);
       End;
    OldINT24[1]:=0;
    OldINT24[2]:=0;
  End;

Function INT24Result: Integer;
  Var
    I:Integer;

  Begin
    I:=IOResult;
    If INT24Err Then
     Begin
      I:=I+256*Succ(INT24ErrCode);
      INT24On;
     End;
    INT24Result:=I;
  End;

{ INT24Result returns all the regular Turbo IOResult codes if no critical
  error has occurred.  If a critical error, then the following values are
  added to the error code from Turbo:
   256:  Attempt to write on write protected disk
   512:  Unknown unit                 [internal dos error]
   768:  Drive not ready              [drive door open or bad drive]
   1024: Unknown command              [internal dos error]
   1280: Data error (CRC)             [bad sector or drive]
   1536: Bad request structure length [internal dos error]
   1792: Seek error                   [bad disk or drive]
   2048: Unknown media type           [bad disk or drive]
   2304: Sector not found             [bad disk or drive]
   2560: Printer out of paper         [anything that the printer might signal]
   2816: Write fault                  [character device not ready]
   3072: Read fault                   [character device not ready]
   3328: General failure              [several meanings]

  If you need the IOResult part, use
   I:=INT24Result and 255; [masks out the INT 24h code]

  For the INT 24h code, use
   I:=INT24Result Shr 8;   [same as Div 256, except faster]

  INT24Result clears both error codes, so you must assign it to a variable if
  you want to extract both codes:
   J:=INT24Result;
   WriteLn('Turbo IOResult  = ',J And 255);
   WriteLn('DOS INT 24h code = ',J Shr 8);

  Note that in most cases, errors on character devices (LST and AUX) will not
  return an IOResult, only an INT 24h error code. }
```

```
{ Main program.  Delete next line to enable }

      {-----------------------------------------------------------}
      {           G E T    E R R O R    C O D E                   }
      {-----------------------------------------------------------}
  Procedure GetErrorCode;
   Begin
   Error := IOresult;                  {Read the I/O result}

   If INT24Err Then
    Begin
     Error:=Error+256*Succ(INT24ErrCode);
     INT24On;
    End;
   Good := (Error = 0);                {Set Boolean Result }
  End;

 {-----------------------------------------------------------------}
```

```
{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
{                        W I N D O . I N C                                }
{                      "...but I dont do floors !"                        }
{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
{                  Kloned and Kludged by Lane Ferris                      }
{                      -- The Hunters Helper --                           }
{                Original Copyright 1984 by Michael A. Covington          }
{                Modifications by Lynn Canning 9/25/85                    }
{                  1) Foreground and Background colors added.             }
{                     Monochrome monitors are automatically set           }
{                     to white on black.                                  }
{                  2) Multiple borders added.                             }
{                  3) TimeDelay procedure added.                          }
{                Requirements: IBM PC or close compatible.                }
{-------------------------------------------------------------------------}
{ To make a window on the screen, call the procedure                      }
{      MkWin(x1,y1,x2,y2,FG,BG,BD);                                       }
{   The x and y coordinates define the window placement and are the       }
{   same as the Turbo Pascal Window coordinates.                          }
{   The border parameters (BD) are 0 = No border                          }
{                                  1 = Single line border                 }
{                                  2 = Double line border                 }
{                                  3 = Double Top/Bottom Single sides     }
{   The foreground (FG) and background (BG) parameters are the same       }
{   values as the corresponding Turbo Pascal values.                      }
{                                                                         }
{ The maximum number of windows open at one time is set at five           }
{ (see MaxWin=5).  This may be set to greater values if necessary.        }
{ After the window is made, you must write the text desired from the      }
{ calling program.  Note that the usable text area is actually 1          }
{ position smaller than the window coordinates to allow for the border.}
{ Hence, a window defined as 1,1,80,25 would actually be 2,2,79,24        }
{ after the border is created.  When writing to the window in your        }
{ calling program, the textcolor and backgroundcolor may be changed as }
{ desired by using the standard Turbo Pascal commands.                    }
{                                                                         }
{ To return to the previous screen or window, call the procedure          }
{      RmWin;                                                             }
{                                                                         }
{ The TimeDelay procedure is invoked from your calling program.  It       }
{ is similar to the Turbo Pascal DELAY except DELAY is based on clock     }
{ speed whereas TimeDelay is based on the actual clock.  This means       }
{ that the delay will be the same duration on all systems no matter       }
{ what the clock speed.                                                   }
{ The procedure could be used for an error condition as follows:          }
{     MkWin          - make an error message window                       }
{     Writeln        - write error message to window                      }
{     TimeDelay(5)   - leave window on screen 5 seconds                   }
{     RmWin          - remove error window                                }
{     cont processing                                                     }
{-------------------------------------------------------------------------}

Const
     InitDone :boolean = false ;      { Initialization switch   }
```

```
      On     = True ;
      Off    = False ;
      VideoEnable = $08;               { Video Signal Enable Bit }
      Bright = 8;                      { Bright Text bit}
      Mono   = 7;                      {MonoChrome Mode}

Type
     Imagetype  = array [1..4000] of char;  { Screen Image in the heap    }
     WinDimtype = record
                    x1,y1,x2,y2: integer
                  end;

     Screens    = record              { Save Screen Information      }
                   Image: Imagetype;  { Saved screen Image }
                   Dim:   WinDimtype; { Saved Window Dimensions }
                   x,y:   integer;    { Saved cursor position }
                  end;


 Var

  Win:                                { Global variable package }
    record
      Dim:    WinDimtype;             { Current Window Dimensions }
      Depth:  integer;
                   { MaxWin should be included in your program }
                   { and it should be the number of windows saved }
                   { at one time }
                   { It should be in the const section of your program }
      Stack:  array[1..MaxWin] of ^Screens;

    end;

  Crtmode     :byte      absolute $0040:$0049; {Crt Mode,Mono,Color,B&W..}
  Crtwidth    :byte      absolute $0040:$004A; {Crt Mode Width, 40:80 .. }
  Monobuffer  :Imagetype absolute $B000:$0000; {Monochrome Adapter Memory}
  Colorbuffer :Imagetype absolute $B800:$0000; {Color Adapter Memory      }
  CrtAdapter  :integer   absolute $0040:$0063; { Current Display Adapter }
  VideoMode   :byte      absolute $0040:$0065; { Video Port Mode byte    }
  TurboCrtMode: byte     absolute  Dseg:6;     {Turbo's Crt Mode byte    }
  Video_Buffer:integer;                        { Record the current Video}
  Delta,
  x,y         :integer;

{-------------------------------------------------------------------}
{                     Delay for  X seconds                          }
{-------------------------------------------------------------------}

procedure TimeDelay (hold : integer);
type
  RegRec =                                { The data to pass to DOS }
    record
      AX, BX, CX, DX, BP, SI, DI, DS, ES, Flags : Integer;
    end;
var
  regs:regrec;
  ah, al, ch, cl, dh:byte;
  sec                 :string[2];
  result, secn, error, secn2, diff :integer;

begin
  ah := $2c;                         {Get Time-Of-Day from DOS}
  with regs do                       {Will give back Ch:hours }
                                     {Cl:minutes,Dh:seconds   }
    ax := ah shl 8 + al;             {Dl:hundreds             }
  intr($21,regs);

  with regs do
    str(dx shr 8:2, sec);            {Get seconds      }
                                     {with leading null}
  if (sec[1] = ' ') then
    sec[1]:= '0';
  val(sec, secn, error);             {Conver seconds to integer}
  repeat                             { stay in this loop until the time }
     ah := $2c;                      { has expired }
     with regs do
        ax := ah shl 8 + al;
     intr($21,regs);                 {Get current time-of-day}

     with regs do                    {Normalize to Char}
        str(dx shr 8:2, sec);
     if (sec[1] = ' ') then
        sec[1]:= '0';
     val(sec, secn2, error);         {Convert seconds to integer}
     diff := secn2 - secn;           {Number of elapsed seconds}
     if diff < 0 then              { we just went over the minute }
        diff := diff + 60;         { so add 60 seconds }
  until diff > hold;               { has our time expired yet }
end; { procedure TimeDelay }

{---------------------------------------------------------------------}
{          Get Absolute postion of Cursor into parameters x,y        }
{---------------------------------------------------------------------}
Procedure Get_Abs_Cursor (var x,y :integer);
  Var
      Active_Page  : byte absolute $0040:$0062;  { Current Video Page Index}
      Crt_Pages    : array[0..7] of integer absolute $0040:$0050 ;

   Begin

      X := Crt_Pages[active_page];      { Get Cursor Position       }
      Y := Hi(X)+1;                     { Y get Row                 }
      X := Lo(X)+1;                     { X gets Col position       }
   End;
{---------------------------------------------------------------------}
{          Turn the Video On/Off to avoid Read/Write snow            }
{---------------------------------------------------------------------}
```

```
Procedure Video (Switch:boolean);
   Begin
      If (Switch = Off) then
      Port[CrtAdapter+4] := (VideoMode - VideoEnable)
      else Port[CrtAdapter+4] := (VideoMode or VideoEnable);
   End;
{------------------------------------------------------------------}
{     InitWin Saves the Current (whole) Screen                     }
{------------------------------------------------------------------}
Procedure InitWin;
  { Records Initial Window Dimensions }
   Begin

     with Win.Dim do
       begin x1:=1; y1:=1; x2:=crtwidth; y2:=25 end;
     Win.Depth:=0;
     InitDone := True ;                  { Show initialization Done }
end;
{------------------------------------------------------------------}
{       BoxWin Draws a Box around the current Window               }
{------------------------------------------------------------------}
procedure BoxWin(x1,y1,x2,y2, BD, FG, BG :integer);

  { Draws a box, fills it with blanks, and makes it the current }
  { Window.  Dimensions given are for the box; actual Window is }
  { one unit smaller in each direction.                         }

var
    I,
    TB,SID,TLC,TRC,BLC,BRC   :integer;

begin
  if Crtmode = Mono then begin
    FG := 7;
    BG := 0;
    end;

  Window(x1,y1,x2,y2);              {Make the Window}
  TextColor(FG) ;                   {Set the colors}
  TextBackground(BG);


  Case BD of                        {Make Border characters}
    0:;                             {No border option}
    1:begin                         {Single line border option}
      TB  := 196;                     {Top Border}
      SID := 179;                     {Side Border}
      TLC := 218;                     {Top Left Corner}
      TRC := 191;                     {Top Right Corner}
      BLC := 192;                     {Bottom Left Corner}
      BRC := 217;                     {Bottom Right Corner}
      end;
    2:begin                         {Double line border option}
      TB  := 205;
      SID := 186;
      TLC := 201; TRC := 187;
      BLC := 200; BRC := 188;
      end;
    3:begin                         {Double Top/Bottom with single sides}
      TB  := 205;                   {"deary and dont spare the lace"}
      SID := 179;
      TLC := 213; TRC := 184;
      BLC := 212; BRC := 190;
      end;
    End;{Case}

  IF BD > 0 then begin                  { User want a border? }
  { Top }
     gotoxy(1,1);                       { Window Origin       }
     Write( chr(TLC) );                 { Top Left Corner     }
     For I:=2 to x2-x1   do             { Top Bar             }
        Write( chr(TB));
     Write( chr(TRC) );                 { Top Right Corner

  { Sides  }
     for I:=2 to y2-y1 do
       begin
         gotoxy(1,I);                   { Left Side Bar       }
         write( chr(SID) );
         gotoxy(x2-x1+1,I) ;            { Right Side Bar      }
         write( chr(SID) );
       end;

  { Bottom }
     gotoxy(1,y2-y1+1);                   { Bottom Left Corner }
     write( chr(BLC) );
     for I:=2 to x2-x1   do               { Bottom Bar         }
        write( chr(TB) );

  { Make it the current Window }
     Window(x1+1,y1+1,x2-1,y2-1);
     write( chr(BRC) );                 { Bottom Right Corner }
   end; {If BD > 0};

   gotoxy(1,1) ;
   TextColor( F6) ;                { Take Low nibble 0..15  }
   TextBackground (B6);            { Take High nibble  0..9 }
   ClrScr;
 end;
{------------------------------------------------------------------}
{       MkWin   Make a Window                                      }
{------------------------------------------------------------------}
procedure MkWin(x1,y1,x2,y2, F6, B6, BD :integer);
  { Create a removable Window }

begin

  If (InitDone = false) then              { Initialize if not done yet }
```

```
    InitWin;

TurboCrtMode := CrtMode;                  {Set Textmode w/o ClrScr}
If CrtMode = 7 then Video_Buffer := $B000 {Set Ptr to Monobuffer        }
else  Video_Buffer := $B800;              {or Color Buffer              }


with Win do Depth:=Depth+1;              { Increment Stack pointer }
if Win.Depth>maxWin then
  begin
    writeln(^G,' Windows nested too deep ');
    halt
  end;
              {-------------------------------------}
              {       Save contents of screen       }
              {-------------------------------------}
With Win do
  Begin
  New(Stack[Depth]);                  { Allocate Current Screen to Heap }
  Video( Off);

  If CrtMode = 7 then
  Stack[Depth]^.Image := monobuffer   { set pointer to it        }
  else
  Stack[Depth]^.Image := colorbuffer ;

  Video( On);
 End ;


With Win do
   Begin                                { Save Screen Dimentions        }
   Stack[Depth]^.Dim := Dim;
   Stack[Win.Depth]^.x  := wherex;      { Save Cursor Position          }
   Stack[Win.Depth]^.y  := wherey;
   End ;

                                        { Validate the Window Placement}
If (X2 > 80) then                       { If off right of screen       }
        begin
        Delta := (X2 - 80);             { Overflow off right margin    }
        If X1 > Delta then
           X1 := X1 - Delta ;           { Move Left window edge        }
        X2 := X2 - Delta ;              { Move Right edge on 80        }
        end;
If (Y2 > 25) then                       { If off bottom   screen       }
        begin
        Delta := Y2 - 25;               { Overflow off right margin    }
        If Y1 > Delta then
           Y1 := Y1 - Delta ;           { Move Top edge up             }
        Y2 := Y2 - Delta ;              { Move Bottom  24              }
        end;
                                        { Create the New Window  }
  BoxWin(x1,y1,x2,y2,BD,FG,BG);
  If BD >0 then begin                    {Shrink window within borders}
     Win.Dim.x1 := x1+1;
     Win.Dim.y1 := y1+1;                   { Allow for margins }
     Win.Dim.x2 := x2-1;
     Win.Dim.y2 := y2-1;
     end;

end;
{---------------------------------------------------------------}
{                        Remove Window                          }
{---------------------------------------------------------------}
        { Remove the most recently created removable Window }
        { Restore screen contents, Window Dimensions, and   }
        { position of cursor.  }
Procedure ReWin;
  Var
    Tempbyte : byte;

   Begin
   Video(Off);

   With Win do
      Begin                            { Restore next Screen       }
      If crtmode = 7 then
      monobuffer := Stack[Depth]^.Image
      else
      colorbuffer := Stack[Depth]^.Image;
      Dispose(Stack[Depth]);            { Remove Screen from Heap   }

   Video(On);

   With Win do                          { Re-instate the Sub-Window }
    Begin                               { Position the old cursor   }
      Dim := Stack[Depth]^.Dim;
      Window(Dim.x1,Dim.y1,Dim.x2,Dim.y2);
      gotoxy(Stack[Depth]^.x,Stack[Depth]^.y);
    end;

      Get_Abs_Cursor(x,y) ;          { New Cursor Position      }
      Tempbyte :=                    { Get old Cursor attributes }
           Mem[ Video_Buffer:((x-1 + (y-1) * 80 ) * 2)+1 ];

      TextColor( Tempbyte And $0F );        { Take Low nibble  0..15}
      TextBackground ( blue );  { Take High nibble  0..9 }
      Depth := Depth - 1
    end ;
end;
{---------------------------------------------------------------}
```

```
{-----------------------------------------------------------}
{            C L O C K _ I 8    Clock Interrupt Service      }
{-----------------------------------------------------------}
(* CLOCK_I8.INL *)
(*  Fm: Neil J. Rubenking [72267,1531]
    On each call to INT 8, this routine checks if the timer is
    "running".  If it is, it checks if the activation time has
    been reached.  If it has, the STATUS byte is set to include
    the "HotKey_On" and "From_Timer" bits.  After that, control
    passes on to the STAYI8.OBJ code *)

  (*NJR*)
  INLINE(
  $9C/                          {PUSHF}
  $2E/$F6/$06/>Status/<Timer_On/ {TEST BY CS:status, timer_on}
  $74/$29/                      {JZ nothing}
  $50/                          {PUSH AX}
  $1E/                          {PUSH DS}
  $B8/$40/$00/                  {MOV AX,40h}
  $8E/$D8/                      {MOV DS,AX}
  $A1/$6E/$00/                  {MOV AX,[6E]}
  $2E/$39/$06/>timer_hi/        {CMP CS:timer_hi,AX}
  $75/$16/                      {JNZ not_yet}
  $A1/$6C/$00/                  {MOV AX,[6C]}
  $2E/$39/$06/>timer_Lo/        {CMP CS:timer_Lo,AX}
  $7D/$0C/                      {JGE Not_Yet}
  $2E/$80/$0E/>Status/<HotKey_On/  {OR BY CS:status, hotkey_on}
  $2E/$80/$0E/>Status/<from_Timer/ {OR BY CS:status, from_timer}

{Not_Yet}
  $1F/                          {POP DS}
  $58/                          {POP AX}
{nothing}
  $9D);                         {POPF}
  (*NJR*)
{------------------- E n d  C l o c k _ I 8 ----------------------}
```

```
Inline(
                           {; STAYI13.400}
                           {;-----------}
                           {; Routine to Set a Flag when INT 13 Disk I/O is active}
  $5D                      {         Pop   Bp                           ; Remove Turbo stack frame}
  /$5D                     {         Pop   Bp}
  /$2E                     {         CS:}
  /$80/$0E/>INTR_FLAGS/<INT13_ON{    Or by [<Intr_flags],<INT13_on      ; Say INT 13 is Active}
  /$9C                     {         Pushf                              ; Invoke Original Disk INT 13}
  /$2E                     {         CS:}
  /$FF/$1E/>BIOS_INT13     {         Call dw [<BIOS_INT13]}
  /$9C                     {         Pushf                              ; Save Return Flags}
  /$2E                     {         CS:}
  /$80/$26/>INTR_FLAGS/<FOXS-INT13_ON{ And by [<Intr_flags],<Foxs-INT13_on; Clear INT 13 Active flag}
  /$9D                     {         Popf                               ; Retrieve results flags}
  /$CA/$02/$00             {         RETf  2                            ; Throw away old flags}
                           {;........................................................................}
);
```

END

12-87

DTIC